

お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願ひ申し上げます。

ルネサスエレクトロニクス ホームページ (<http://www.renesas.com>)

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社 (<http://www.renesas.com>)

【問い合わせ先】<http://japan.renesas.com/inquiry>

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事事務の目的で使用しないでください。また、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に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したのですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。
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
高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）
特定水準： 航空機器、航空宇宙機器、海中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関し、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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とその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

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

お客様各位

資料中の「日立製作所」、「日立XX」等名称の株式会社ルネサス テクノロジへの変更について

2003年4月1日を以って三菱電機株式会社及び株式会社日立製作所のマイコン、ロジック、アナログ、ディスクリット半導体、及びDRAMを除くメモリ(フラッシュメモリ・SRAM等)を含む半導体事業は株式会社ルネサス テクノロジに承継されました。従いまして、本資料中には「日立製作所」、「株式会社日立製作所」、「日立半導体」、「日立XX」といった表記が残っておりますが、これらの表記は全て「株式会社ルネサス テクノロジ」に変更されておりますのでご理解の程お願い致します。尚、会社商標・ロゴ・コーポレートステートメント以外の内容については一切変更しておりませんので資料としての内容更新ではありません。

ルネサステクノロジ ホームページ (<http://www.renesas.com>)

2003年4月1日
株式会社ルネサス テクノロジ
カスタマサポート部

ご注意

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご注意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりますとは、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ (<http://www.renesas.com>)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものです。万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任を負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気づきの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

SuperH RISC engine
C/C++コンパイラ、アセンブラ、
最適化リンケージエディタ

ユーザーズマニュアル

ルネサスマイクロコンピュータ開発環境システム

HS0700CLCS7SJ

ご注意

- 1 本書に記載の製品及び技術のうち「外国為替及び外国貿易法」に基づき安全保障貿易管理関連貨物・技術に該当するものを輸出する場合、または国外に持ち出す場合は日本国政府の許可が必要です。
- 2 本書に記載された情報の使用に際して、弊社もしくは第三者の特許権、著作権、商標権、その他の知的所有権等の権利に対する保証または実施権の許諾を行うものではありません。また本書に記載された情報を使用した事により第三者の知的所有権等の権利に関わる問題が生じた場合、弊社はその責を負いませんので予めご了承ください。
- 3 製品及び製品仕様は予告無く変更する場合がありますので、最終的な設計、ご購入、ご使用に際しましては、事前に最新の製品規格または仕様書をお求めになりご確認ください。
- 4 弊社は品質・信頼性の向上に努めておりますが、宇宙、航空、原子力、燃焼制御、運輸、交通、各種安全装置、ライフサポート関連の医療機器等のように、特別な品質・信頼性が要求され、その故障や誤動作が直接人命を脅かしたり、人体に危害を及ぼす恐れのある用途にご使用をお考えのお客様は、事前に弊社営業担当迄ご相談をお願い致します。
- 5 設計に際しては、特に最大定格、動作電源電圧範囲、放熱特性、実装条件及びその他諸条件につきましては、弊社保証範囲内でご使用いただきますようお願い致します。
保証値を越えてご使用された場合の故障及び事故につきましては、弊社はその責を負いません。また保証値内のご使用であっても半導体製品について通常予測される故障発生率、故障モードをご考慮の上、弊社製品の動作が原因でご使用機器が人身事故、火災事故、その他の拡大損害を生じないようにフェールセーフ等のシステム上の対策を講じて頂きますようお願い致します。
- 6 本製品は耐放射線設計をしておりません。
- 7 本書の一部または全部を弊社の文書による承認なしに転載または複製することを堅くお断り致します。
- 8 本書をはじめ弊社半導体についてのお問い合わせ、ご相談は弊社営業担当迄お願い致します。

はじめに

本マニュアルは、「SuperH RISC engine C/C++コンパイラ、アセンブラ、最適化リンケージエディタ」の使用方法を述べたものです。

本製品は C 言語、C++ 言語およびアセンブリ言語で記述したソースプログラムを、SuperH RISC engine 用オブジェクトプログラムおよびロードモジュールに変換するソフトウェアシステムです。ご使用になる前に、本マニュアルを良く読んで理解してください。

表記上の注意事項

本マニュアルの説明の中で用いられる記号は、次の意味を示しています。

- この記号で囲まれた内容を指定することを示します。
- [] 省略してもよい項目を示します。
- . . . 直前の項目を 1 回以上指定することを示します。
1 個以上の空白を示します。
- (RET) キャリッジリターンキー(リターンキーともいいます)を示します。
- | | で区切られた項目を選択できることを示します。
- (CNTL) 次の文字を、コントロールキーを押しながら入力することを示します。

本マニュアルは UNIX^{*1}、または PC-9801^{*2} シリーズ、IBM PC^{*3} およびその互換機上で動作する Microsoft® Windows® 95 operating system、Microsoft® Windows® 98 operating system、Microsoft® Windows® Millennium Edition operating system、Microsoft® Windows NT® operating system、Microsoft® Windows® 2000 operating system^{*4} に対応するように書かれています。UNIX 上で動作するソフトウェアシステムを以下 UNIX 版と称します。または PC-9801 シリーズ、IBM PC およびその互換機上で動作するソフトウェアシステムを以下 PC 版と称します。

- 【注】*1 UNIX は、X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標です。
- *2 PC-9801 は、日本電気株式会社の商標です。
- *3 IBM PC は、米国 International Business Machines Corporation の登録商標です。
- *4 Microsoft®, Windows®, Windows NT®は、米国 Microsoft Corporation の米国及びその他の国における登録商標または商標です。

目次

1. 概要	1
1.1 プログラムの開発手順	1
1.2 コンパイラの概要	2
1.3 アセンブラの概要	2
1.4 最適化リンケージエディタの概要	2
1.5 プレリンカの概要	3
1.6 標準ライブラリ構築ツールの概要	3
1.7 スタック解析ツールの概要	3
1.8 フォーマットコンバータの概要	3
2. C/C++コンパイラ操作方法	5
2.1 オプション指定規則	5
2.2 オプション解説	5
2.2.1 Source オプション	5
<i>Include</i>	6
<i>PREInclude</i>	6
<i>DEFine</i>	7
<i>MESsage</i>	7
<i>NOMESsage</i>	7
2.2.2 Object オプション	8
<i>PREProcessor</i>	9
<i>Code</i>	9
<i>DEBug</i>	10
<i>NODEBug</i>	10
<i>SEction</i>	10
<i>SString</i>	11
<i>OBjectfile</i>	11
<i>Template</i>	12
<i>ABs16</i>	12
<i>Division</i>	13
2.2.3 List オプション	14
<i>Listfile</i>	14
<i>NOListfile</i>	14
<i>SHow</i>	15
2.2.4 Optimize オプション	16
<i>OPTimize</i>	16
<i>SPEED</i>	17
<i>SIze</i>	17

	<i>NOSpeed</i>	17
	<i>Goptimize</i>	17
	<i>MAP</i>	18
	<i>GB r</i>	20
	<i>CAsE</i>	21
	<i>SHft</i>	21
	<i>BLOckcopy</i>	22
	<i>INLine</i>	22
	<i>NOINLine</i>	22
2.2.5	Other オプション	23
	<i>ECpp</i>	24
	<i>COMment</i>	24
	<i>Macsave</i>	25
	<i>ALign16</i>	25
	<i>NOALign16</i>	25
	<i>RTnext</i>	26
	<i>NORTnext</i>	26
	<i>LOop</i>	26
	<i>NOLOop</i>	26
	<i>APproxdiv</i>	27
	<i>PAth</i>	27
	<i>FPScr</i>	28
	<i>Volatile_loop</i>	28
2.2.6	CPU オプション	29
	<i>CPu</i>	30
	<i>ENdian</i>	30
	<i>FPu</i>	31
	<i>Round</i>	31
	<i>DENormalize</i>	32
	<i>Pic</i>	32
	<i>DOuble = Float</i>	33
	<i>EXception</i>	33
	<i>NOEXception</i>	33
	<i>RTTI</i>	34
2.2.7	その他オプション	35
	<i>LAng</i>	35
	<i>LOGO</i>	36
	<i>NOLOGO</i>	36
	<i>Euc</i>	36
	<i>SJis</i>	36
	<i>LATin1</i>	36
	<i>OUnicode</i>	37
	<i>SUBcommand</i>	37
3.	アセンブラオプション	39
3.1	オプション指定規則	39
3.2	オプション解説	39

3.2.1	Source オプション	39
	<i>Include</i>	40
	<i>DEFine</i>	40
	<i>ASsignA</i>	41
	<i>ASsignC</i>	42
3.2.2	Object オプション	43
	<i>Debug</i>	43
	<i>NODEbug</i>	43
	<i>EXPand</i>	44
	<i>LITERAL</i>	44
	<i>Object</i>	45
	<i>NOObject</i>	45
3.2.3	List オプション	46
	<i>LISt</i>	47
	<i>NOLISt</i>	47
	<i>SOurce</i>	48
	<i>NOSource</i>	48
	<i>SHow</i>	49
	<i>NOSHow</i>	49
	<i>CRoss_reference</i>	50
	<i>NOCross_reference</i>	50
	<i>SEction</i>	51
	<i>NOSEction</i>	51
3.2.4	Other オプション	52
	<i>AUTO_literal</i>	52
3.2.5	CPU オプション	53
	<i>CPU</i>	53
	<i>ENdian</i>	54
	<i>Round</i>	54
	<i>DENormalize</i>	55
3.2.6	その他のオプション	56
	<i>ABort</i>	56
	<i>LATINI</i>	57
	<i>SJIS</i>	57
	<i>EUC</i>	58
	<i>OUnicode</i>	58
	<i>LINes</i>	59
	<i>COlumnS</i>	59
	<i>LOGO</i>	60
	<i>NOLOGO</i>	60
	<i>SUBcommand</i>	60
4.	最適化リンケージエディタ操作方法	61
4.1	オプション指定規則	61
4.1.1	コマンドラインの形式	61
4.1.2	サブコマンドファイルの形式	61

4.2 オプション解説	62
4.2.1 Input オプション	62
Input	63
LIBrary	63
Binary	64
DEFine	64
ENTry	65
NOPRElink	65
4.2.2 Output オプション	66
FOrm	67
DEBug	69
SDebug	69
NODEBug	69
REcord	69
ROm	70
OUtput	70
Message	71
NOMessage	71
MAP	71
4.2.3 List オプション	72
LISt	72
SHow	73
4.2.4 Optimize オプション	74
OPtimize	75
NOOPtimize	75
SAMESize	76
PROfile	76
CAchesize	77
SYmbol_forbid	77
SAMECode_forbid	77
Variable_forbid	77
FUnction_forbid	77
Absolute_forbid	77
4.2.5 Section オプション	78
STARt	78
FSymbol	79
4.2.6 Verify オプション	80
CPu	80
4.2.7 Other オプション	81
S9	82
STACK	82
Cmpress	83
NOCmpress	83
REName	83
DELete	84
REPlace	84
EXTRact	85

<i>STRip</i> _____	85
<i>CHange_message</i> _____	86
<i>LOgo</i> _____	86
<i>NOLOgo</i> _____	86
<i>END</i> _____	87
<i>EXIt</i> _____	87
4.2.8 subcommand file オプション _____	88
<i>SUbcmand</i> _____	88
5. 標準ライブラリ構築ツール操作方法 _____	89
5.1 オプション指定規則 _____	89
5.2 オプション解説 _____	89
5.2.1 追加オプション _____	89
<i>Head</i> _____	90
<i>OUTPut</i> _____	90
<i>NOFLoat</i> _____	91
<i>REent</i> _____	91
5.2.2 指定不可オプション _____	92
5.2.3 オプション指定時の注意事項 _____	92
6. スタック解析ツール操作方法 _____	93
6.1 スタック解析ツールの起動 _____	93
6.2 スタック解析ツールの機能概要 _____	94
7. 環境変数 _____	97
7.1 環境変数一覧 _____	97
7.2 コンパイラの暗黙の宣言 _____	100
8. ファイル仕様 _____	101
8.1 ファイル名の付け方 _____	101
8.2 コンパイルリストの参照方法 _____	102
8.2.1 コンパイルリストの構成 _____	102
8.2.2 ソースリスト情報 _____	102
8.2.3 オブジェクト情報 _____	105
8.2.4 統計情報 _____	107
8.2.5 コマンド指定情報 _____	108
8.3 アセンブルリストの参照方法 _____	109
8.3.1 アセンブルリストの構成 _____	109
8.3.2 ソースリスト情報 _____	110
8.3.3 クロスリファレンスリスト _____	111
8.3.4 セクション情報リスト _____	112
8.4 リンケージリストの参照方法 _____	113
8.4.1 リンケージリストの構成 _____	113
8.4.2 オプション情報 _____	113

8.4.3	エラー情報	114
8.4.4	リンケージマップ情報	114
8.4.5	シンボル情報	115
8.4.6	シンボル削除最適化情報	116
8.4.7	変数アクセス最適化対象シンボル情報	116
8.4.8	関数アクセス最適化対象シンボル情報	116
8.5	ライブラリリストの参照方法	117
8.5.1	ライブラリリストの構成	117
8.5.2	オプション情報	117
8.5.3	エラー情報	118
8.5.4	ライブラリ情報	118
8.5.5	ライブラリ内モジュール、セクション、シンボル情報	119
9.	プログラミング	121
9.1	プログラムの構造	121
9.1.1	セクション	121
9.1.2	C/C++プログラムのセクション	121
9.1.3	アセンブリプログラムのセクション	124
9.1.4	セクションの結合	125
9.2	初期設定プログラムの作成	128
9.2.1	メモリ領域の割り付け	129
9.2.2	実行環境の設定	136
	(ルーチン名)	149
	<i>int open(char *name, int mode, int flg)</i>	150
	<i>int close(int fileno)</i>	151
	<i>int read(int fileno, char *buf, unsigned int count)</i>	151
	<i>int write(int fileno, char *buf, unsigned int count)</i>	152
	<i>int lseek(int fileno, long offset, int base)</i>	152
	<i>char *sbrk(int size)</i>	153
	<i>int *errno_addr(void)</i>	153
	<i>int wait_sem (int semnum)</i>	154
	<i>int signal_sem (int semnum)</i>	154
9.3	C/C++プログラムとアセンブリプログラムとの結合	165
9.3.1	外部名の相互参照方法	165
9.3.2	関数呼び出しのインタフェース	167
9.3.3	引数割り付けの具体例	174
9.3.4	レジスタとスタック領域の使用法	176
9.4	プログラム作成上の注意事項	177
9.4.1	コーディング上の注意事項	177
9.4.2	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	179
9.4.3	プログラム開発上の注意事項	180
10.	C/C++言語仕様	181
10.1	言語仕様	181

10.1.1	コンパイラの仕様	181
10.1.2	データの内部表現	186
10.1.3	浮動小数点数の仕様	196
10.1.4	演算子の評価順序	202
10.2	拡張機能	203
10.2.1	#pragma 拡張子	203
	#pragma section	204
	#pragma abs16	205
	#pragma stacksize	205
	#pragma interrupt	206
	#pragma inline	209
	#pragma inline_asm	210
	#pragma regsave	212
	#pragma noregsave	212
	#pragma noregalloc	212
	#pragma entry	213
	#pragma global_register	214
	#pragma gbr_base	215
	#pragma gbr_base1	215
10.2.2	セクションアドレス演算子	216
	__sectop	216
	__secend	216
10.2.3	組み込み関数	217
	void set_cr(int cr)	219
	int get_cr(void)	219
	void set_imask(int mask)	220
	int get_imask(void)	220
	void set_vbr(void *base)	221
	void *get_vbr(void)	221
	void set_gbr(void *base)	222
	void *get_gbr(void)	222
	unsigned char gbr_read_byte(int offset)	223
	unsigned short gbr_read_word(int offset)	223
	unsigned long gbr_read_long(int offset)	224
	void gbr_write_byte(int offset, unsigned char data)	224
	void gbr_write_word(int offset, unsigned short data)	225
	void gbr_write_long(int offset, unsigned long data)	225
	void gbr_and_byte(int offset, unsigned char mask)	226
	void gbr_or_byte(int offset, unsigned char mask)	226
	void gbr_xor_byte(int offset, unsigned char mask)	227
	int gbr_tst_byte(int offset, unsigned char mask)	227
	void sleep(void)	229
	int tas(char *addr)	229
	int trapa(int trap_no)	230
	int trapa_svc(int trap_no, int code, type1 para1, type2 para2, type3 para3, type4 para4)	230
	void prefetch(void *p)	231

<i>void trace (long v)</i>	231
<i>void ldilb (void)</i>	232
<i>void nop (void)</i>	232
<i>long dmuls_h (long data1, long data2)</i>	233
<i>unsigned long dmuls_l (long data1, long data2)</i>	233
<i>unsigned long dmulu_h (unsigned long data1, unsigned long data2)</i>	234
<i>unsigned long dmulu_l (unsigned long data1, unsigned long data2)</i>	234
<i>unsigned short swapb(unsigned short data)</i>	235
<i>unsigned long swapw(unsigned long data)</i>	235
<i>unsigned long end_cnvf(unsigned long data)</i>	236
<i>int macw(short *ptr1, short *ptr2, unsigned int count)</i>	236
<i>int macwl(short *ptr1, short *ptr2, unsigned int count, unsigned int mask)</i>	236
<i>int macl(int *ptr1, int *ptr2, unsigned int count)</i>	237
<i>int macll(int *ptr1, int *ptr2, unsigned int count, unsigned int mask)</i>	237
<i>void set_fpscr(int cr)</i>	237
<i>int get_fpscr(void)</i>	238
<i>float fipr(float vect1[4], float vect2[4])</i>	238
<i>void firv(float vec1[4], float vec2[4])</i>	239
<i>void firvadd(float vec1[4], float vec2[4], float vec3[4])</i>	240
<i>void firvsub(float vec1[4], float vec2[4], float vec3[4])</i>	241
<i>void add4(float vec1[4], float vec2[4], float vec3[4])</i>	242
<i>void sub4(float vec1[4], float vec2[4], float vec3[4])</i>	242
<i>void mtrx4mul(float mat1[4], float mat2[4])</i>	243
<i>void mtrx4muladd(float mat1[4], float mat2[4], float mat3[4])</i>	244
<i>void mtrx4mulsub(float mat1[4], float mat2[4], float mat3[4])</i>	244
<i>void ld_ext(float mat[4][4])</i>	245
<i>void st_ext(float mat[4][4])</i>	245
10.3C/C++ライブラリ	246
10.3.1 標準Cライブラリ	246
<i>void assert(int expression)</i>	253
<i>int isalnum(int c)</i>	255
<i>int isalpha(int c)</i>	256
<i>int iscntrl(int c)</i>	256
<i>int isdigit(int c)</i>	257
<i>int isgraph(int c)</i>	257
<i>int islower(int c)</i>	258
<i>int isprint(int c)</i>	258
<i>int ispunct(int c)</i>	259
<i>int isspace(int c)</i>	259
<i>int isupper(int c)</i>	260
<i>int isxdigit(int c)</i>	260
<i>int tolower(int c)</i>	261
<i>int toupper(int c)</i>	261
<i>double acos(double d)</i>	269
<i>double asin(double d)</i>	269
<i>double atan(double d)</i>	270
<i>double atan2(double y, double x)</i>	271

<i>double cos(double d)</i>	272
<i>double sin(double d)</i>	272
<i>double tan(double d)</i>	273
<i>double cosh(double d)</i>	273
<i>double sinh(double d)</i>	274
<i>double tanh(double d)</i>	274
<i>double exp(double d)</i>	275
<i>double frexp(double value, int *e)</i>	275
<i>double ldexp(double e, int f)</i>	276
<i>double log(double d)</i>	276
<i>double log10(double d)</i>	277
<i>double modf(double a, double *b)</i>	277
<i>double pow(double x, double y)</i>	278
<i>double sqrt(double d)</i>	278
<i>double ceil(double d)</i>	279
<i>double fabs(double d)</i>	279
<i>double floor(double d)</i>	280
<i>double fmod(double x, double y)</i>	280
<i>float acosf(float f)</i>	283
<i>float asinf(float f)</i>	283
<i>float atanf(float f)</i>	284
<i>float atan2f(float y, float x)</i>	285
<i>float cosf(float f)</i>	286
<i>float sinf(float f)</i>	286
<i>float tanf(float f)</i>	287
<i>float coshf(float f)</i>	287
<i>float sinh(float f)</i>	288
<i>float tanhf(float f)</i>	288
<i>float expf(float f)</i>	289
<i>float frexpf(float value, int *e)</i>	289
<i>float ldexpf(float e, int f)</i>	290
<i>float logf(float f)</i>	290
<i>float log10f(float f)</i>	291
<i>float modff(float a, float *b)</i>	291
<i>float powf(float x, float y)</i>	292
<i>float sqrtf(float f)</i>	292
<i>float ceilf(float f)</i>	293
<i>float fabsf(float f)</i>	293
<i>float floorf(float f)</i>	294
<i>float fmodf(float x, float y)</i>	294
<i>int setjmp(jmp_buf env)</i>	296
<i>void longjmp(jmp_buf env, int ret)</i>	296
<i>void va_start(va_list ap, parmN)</i>	298
<i>type va_arg(va_list ap, type)</i>	298
<i>void va_end(va_list ap)</i>	299
<i>int fclose(FILE *fp)</i>	304
<i>int fflush(FILE *fp)</i>	304

<i>FILE *fopen(const char *fname, const char *mode)</i>	305
<i>FILE *freopen(const char *fname, const char *mode, FILE *fp)</i>	306
<i>void setbuf(FILE *fp, char buf[BUFSIZ])</i>	307
<i>int servbuf(FILE *fp, char *buf, int type, size_t size)</i>	308
<i>int fprintf(FILE *fp, const char *control [, arg ...])</i>	309
<i>int fscanf(FILE *fp, const char *control [, ptr ...])</i>	314
<i>int printf(const char *control [, arg ...])</i>	317
<i>int scanf(const char *control [, ptr ...])</i>	317
<i>int sprintf(char *s, const char *control [, arg ...])</i>	318
<i>int sscanf(const char *s, const char *control [, ptr ...])</i>	318
<i>int vfprintf(FILE *fp, const char *control, va_list arg)</i>	319
<i>int vprintf(const char *control, va_list arg)</i>	320
<i>int vsprintf(char *s, const char *control, va_list arg)</i>	321
<i>int fgetc(FILE *fp)</i>	322
<i>char *fgets(char *s, int n, FILE *fp)</i>	323
<i>int fputc(int c, FILE *fp)</i>	324
<i>int fputs(const char *s, FILE *fp)</i>	325
<i>int getc(FILE *fp)</i>	326
<i>int getchar(void)</i>	326
<i>char *gets(char *s)</i>	327
<i>int putc(int c, FILE *fp)</i>	327
<i>int putchar(int c)</i>	328
<i>int puts(const char *s)</i>	328
<i>int ungetc(int c, FILE *fp)</i>	329
<i>size_t fread(void *ptr, size_t size, size_t n, FILE *fp)</i>	330
<i>size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp)</i>	331
<i>int fseek(FILE *fp, long offset, int type)</i>	332
<i>long ftell(FILE *fp)</i>	333
<i>void rewind(FILE *fp)</i>	333
<i>void clearerr(FILE *fp)</i>	334
<i>int feof(FILE *fp)</i>	334
<i>int ferror(FILE *fp)</i>	335
<i>void perror(const char *s)</i>	335
<i>double atof(const char *nptr)</i>	337
<i>int atoi(const char *nptr)</i>	337
<i>long atol(const char *nptr)</i>	338
<i>double strtod(const char *nptr, char **endptr)</i>	339
<i>long strtol(const char *nptr, char **endptr, int base)</i>	340
<i>int rand(void)</i>	341
<i>void srand(unsigned int seed)</i>	341
<i>void *calloc(size_t nelem, size_t elsize)</i>	342
<i>void free(void *ptr)</i>	342
<i>void *malloc(size_t size)</i>	343
<i>void *realloc(void *ptr, size_t size)</i>	343
<i>void *bsearch(const void *key, const void *base, size_t nmemb, size_t size, _____</i>	344
<i>int (*compar)(const void *, const void *)</i>	344
<i>void qsort(const void *base, size_t nmemb, size_t size, _____</i>	345

<i>int (*compar)(const void *, const void *)</i>	345
<i>int abs(int i)</i>	345
<i>div_t div(int numer, int denom)</i>	346
<i>long labs(long j)</i>	346
<i>ldiv_t ldiv(long numer, long denom)</i>	347
<i>void *memcpy(void *s1, const void *s2, size_t n)</i>	350
<i>char *strcpy(char *s1, const char *s2)</i>	350
<i>char *strncpy(char *s1, const char *s2, size_t n)</i>	351
<i>char *strcat(char *s1, const char *s2)</i>	351
<i>char *strncat(char *s1, const char *s2, size_t n)</i>	352
<i>int memcmp(const void *s1, const void *s2, size_t n)</i>	352
<i>int strcmp(const char *s1, const char *s2)</i>	353
<i>int strncmp(const char *s1, const char *s2, size_t n)</i>	353
<i>void *memchr(const void *s, int c, size_t n)</i>	354
<i>char *strchr(const char *s, int c)</i>	354
<i>size_t strcspn(const char *s1, const char *s2)</i>	355
<i>char *strpbrk(const char *s1, const char *s2)</i>	355
<i>char *strrchr(const char *s, int c)</i>	356
<i>size_t strspn(const char *s1, const char *s2)</i>	356
<i>char *strstr(const char *s1, const char *s2)</i>	357
<i>char *strtok(char *s1, const char *s2)</i>	358
<i>void *memset(void *s, int c, size_t n)</i>	359
<i>char *strerror(int s)</i>	359
<i>size_t strlen(const char *s)</i>	360
<i>void *memmove(void *s1, const void *s2, size_t n)</i>	360
10.3.2 EC++クラスライブラリ	361
10.3.3 リエントラントライブラリ	432
10.3.4 未サポートライブラリ	433
10.3.5 DSP ライブラリ	434
<i>int FftComplex (short op_x[], short op_y[], const short ip_x[],</i> <i>const short ip_y[], long size, long scale)</i>	439
<i>int FftReal (short op_x[], short op_y[], const short ip[], long size, long scale)</i>	440
<i>int IfftComplex (short op_x[], short op_y[], const short ip_x[],</i> <i>const short ip_y[], long size, long scale)</i>	441
<i>int IfftReal (short op_x[], short scratch_y[], const short ip_x[],</i> <i>const short ip_y[], long size, long scale, int op_all_x)</i>	442
<i>int FftInComplex (short data_x[], short data_y[], long size, long scale)</i>	443
<i>int FftInReal (short data_x[], short data_y[], long size, long scale, int ip_all_x)</i>	444
<i>int IfftInComplex (short data_x[], short data_y[], long size, long scale)</i>	445
<i>int IfftInReal (short data_x[], short data_y[], long size, long scale, int op_all_x)</i>	446
<i>int LogMagnitude (short output[], const short ip_x[], const short ip_y[],</i> <i>long no_elements, float fscale)</i>	447
<i>int InitFft (long max_size)</i>	448
<i>void FreeFft (void)</i>	448
<i>int GenBlackman (short output[], long win_size)</i>	449
<i>int GenHamming (short output[], long win_size)</i>	450
<i>int GenHanning (short output[], long win_size)</i>	450

<i>int GenTriangle (short output[], long win_size)</i>	451
<i>int Fir (short output[], const short input[], long no_samples, const short coeff[], long no_coefs, int res_shift, short *workspace)</i>	454
<i>int Fir1 (short *output, short input, const short coeff[], long no_coefs, int res_shift, short *workspace)</i>	455
<i>int Iir (short output[], const short input[], long no_samples, const short coeff[], long no_sections, short *workspace)</i>	456
<i>int Iir1 (short *output, short input, const short coeff[], long no_sections, short *workspace)</i>	457
<i>int Dfir (short output[], const short input[], long no_samples, const long coeff[], long no_sections, long *workspace)</i>	458
<i>int Dfir1 (short *output, const short input, const long coeff[], long no_sections, long *workspace)</i>	459
<i>int Lms (short output[], const short input[], const short ref_output[], long no_samples, short coeff[], long no_coefs, int res_shift, short conv_fact, short *workspace)</i>	460
<i>int Lms1 (short *output, short input, short ref_output, short coeff[], long no_coefs, int res_shift, short conv_fact, short *workspace)</i>	461
<i>int InitFir (short **workspace, long no_coefs)</i>	462
<i>int InitIir (short **workspace, long no_sections)</i>	462
<i>int InitDfir (long **workspace, long no_sections)</i>	463
<i>int InitLms (short **workspace, long no_coefs)</i>	463
<i>int FreeFir (short **workspace, long no_coefs)</i>	464
<i>int FreeIir (short **workspace, long no_sections)</i>	464
<i>int FreeDfir (long **workspace, long no_sections)</i>	465
<i>int FreeLms (short **workspace, long no_coefs)</i>	465
<i>int ConvComplete (short output[], const short ip_x[], const short ip_y[], long x_size, long y_size, int res_shift)</i>	466
<i>int ConvCyclic (short output[], const short ip_x[], const short ip_y[], long size, int res_shift)</i>	467
<i>int ConvPartial (short output[], const short ip_x[], const short ip_y[], long x_size, long y_size, int res_shift)</i>	468
<i>int Correlate (short output[], const short ip_x[], const short ip_y[], long x_size, long y_size, long no_corr, int x_is_larger, int res_shift)</i>	469
<i>int CorrCyclic (short output[], const short ip_x[], const short ip_y[], long size, int reverse, int res_shift)</i>	470
<i>int Limit (short data_xy[], long no_elements, int data_is_x)</i>	471
<i>int CopyXtoY (short op_y[], const short ip_x[], long n)</i>	472
<i>int CopyYtoX (short op_x[], const short ip_y[], long n)</i>	472
<i>int CopyToX (short op_x[], const short input[], long n)</i>	473
<i>int CopyToY (short op_y[], const short input[], long n)</i>	473
<i>int CopyFromX (short output[], const short ip_x[], long n)</i>	474
<i>int CopyFromY (short output[], const short ip_y[], long n)</i>	474
<i>int GenGnoise (short output[], long no_samples, float variance)</i>	475
<i>int MatrixMult (void *op_matrix, const void *ip_x, const void *ip_y, long m, long n, long p, int x_first, int res_shift)</i>	476
<i>int VectorMult (short output[], const short ip_x[], const short ip_y[],</i>	477

<i>long no_elements, int res_shift)</i>	477
<i>int MsPower (long *output, const short input[], long no_elements, int src_is_x)</i>	477
<i>int Mean (short *mean, const short input[], long no_elements, int src_is_x)</i>	478
<i>int Variance (long *variance, short *mean, const short input[],</i>	479
<i>long no_elements, int src_is_x)</i>	479
<i>int MaxI (short **max_ptr, short input[], long no_elements, int src_is_x)</i>	480
<i>int MinI (short **min_ptr, short input[], long no_elements, int src_is_x)</i>	480
<i>int PeakI (short **peak_ptr, short input[], long no_elements, int src_is_x)</i>	481
11. アセンブラ言語仕様	483
11.1 プログラムの要素	483
11.1.1 ソースステートメント	483
11.1.2 予約語	485
11.1.3 シンボル	486
11.1.4 定数488	
11.1.5 ロケーションカウンタ	494
11.1.6 式	495
11.1.7 文字列	501
11.1.8 ローカルラベル	501
11.2 実行命令	503
11.2.1 実行命令の概要	503
11.2.2 実行命令に関する注意事項	507
11.3 DSP 命令	523
11.3.1 プログラムの要素	523
11.3.2 DSP 命令	526
11.4 アセンブラ制御命令	532
.CPU	533
.SECTION	534
.ORG	537
.ALIGN	538
.EQU	539
.ASSIGN	540
.REG	541
.FREG	542
.DATA	543
.DATAB	544
.SDATA	545
.SDATAB	546
.SDATAC	547
.SDATAZ	548
.FDATA	549
.FDATAB	550
.XDATA	551
.RES	552
.SRES	553
.SRESC	554

.SRESZ	555
.FRES	556
.EXPORT	557
.IMPORT	558
.GLOBAL	559
.OUTPUT	560
.DEBUG	562
.ENDIAN	563
.LINE	565
.PRINT	566
.LIST	567
.FORM	570
.HEADING	571
.PAGE	572
.SPACE	573
.PROGRAM	574
.RADIX	575
.END	576
11.5 ファイルインクルード機能	577
11.6 条件つきアセンブリ機能	579
11.6.1 条件つきアセンブリ機能の概要	579
11.6.2 条件つきアセンブリ機能に関する制御文	583
.ASSIGNA	584
.ASSIGNC	586
.DEFINE	587
.AIF, .AELIF, .AELSE, .AENDI	588
.AIFDEF, .AELSE, .AENDI	590
.AREPEAT, .AENDR	591
.AWHILE, .AENDW	592
.EXITM	593
.AERROR	594
.ALIMIT	594
11.7 マクロ機能	595
11.7.1 マクロ機能の概要	595
11.7.2 マクロ機能に関する制御文	597
.MACRO, .ENDM	598
11.7.3 マクロ本体	600
11.7.4 マクロコール	603
11.7.5 文字列操作関数	605
.LEN	605
.INSTR	606
.SUBSTR	607
11.8 リテラルプールの自動生成機能	608
11.8.1 リテラルプールの自動生成機能の概要	608
11.8.2 リテラルプールの自動生成機能に関する拡張命令	608
11.8.3 リテラルプールの自動生成機能のサイズモード	609

11.8.4	リテラルプールの出力	610
11.8.5	リテラルの共有	612
11.8.6	リテラルプール出力の抑止	613
11.8.7	リテラルプール自動生成に関する注意事項	614
11.9	リピートループ命令自動生成機能	616
11.9.1	リピートループ命令自動生成機能の概要	616
11.9.2	リピートループ命令自動生成機能に関する拡張命令	617
11.9.3	REPEAT の記述方法	617
11.9.4	コーディング例	618
11.9.5	REPEAT 拡張命令に関する注意事項	620
12.	コンパイラのエラーメッセージ	621
12.1	エラー形式とエラーレベル	621
12.2	メッセージ一覧	621
12.3	標準ライブラリのエラーメッセージ	684
13.	アセンブラのエラーメッセージ	687
13.1	エラー形式とエラーレベル	687
13.2	メッセージ一覧	687
14.	最適化リンケージエディタのエラーメッセージ	707
14.1	エラー形式とエラーレベル	707
14.2	メッセージ一覧	707
15.	標準ライブラリ構築ツール・フォーマットコンバータの エラーメッセージ	719
15.1	エラー形式とエラーレベル	719
15.2	メッセージ一覧	719
16.	限界値	721
16.1	コンパイラの限界値	721
16.2	アセンブラの限界値	722
17.	バージョンアップにおける注意事項	723
17.1	バージョンアップ時の注意事項	723
17.1.1	プログラムの動作保証	723
17.1.2	旧バージョンとの互換性	724
17.1.3	コマンドラインインタフェース	725
17.1.4	提供内容	727
17.1.5	リストファイル仕様	727
17.2	追加・改善内容	727
17.2.1	共通の追加・改善(パッケージ Ver.6)	727
17.2.2	コンパイラの追加・改善機能	728

17.2.3 最適化リンケージエディタの追加・改善機能	730
17.3 フォーマットコンバータ操作方法	731
17.3.1 オブジェクトファイル形式	731
17.3.2 旧バージョンとの互換性	731
17.3.3 オプション指定規則	732
17.3.4 オプション解説	732
<i>Fpu</i>	733
<i>Dsp</i>	733
<i>Endian</i>	734
<i>Sysrof</i>	735
<i>Dwarf1</i>	735
18. 付録	737
18.1 S タイプ、HEX ファイル形式	737
18.1.1 S タイプファイル形式	737
18.1.2 HEX ファイル形式	739
18.2 ASCII コード一覧表	741

1. 概要

1.1 プログラムの開発手順

プログラムの開発手順を図 1.1 に示します。網掛け部分は、「SuperH RISC engine C/C++コンパイラパッケージ」として提供するソフトウェアを示します。

本マニュアルでは、C/C++コンパイラ、アセンブラ、最適化リンケージエディタ、標準ライブラリ構築ツール、スタック解析ツール、フォーマットコンバータについて説明します。

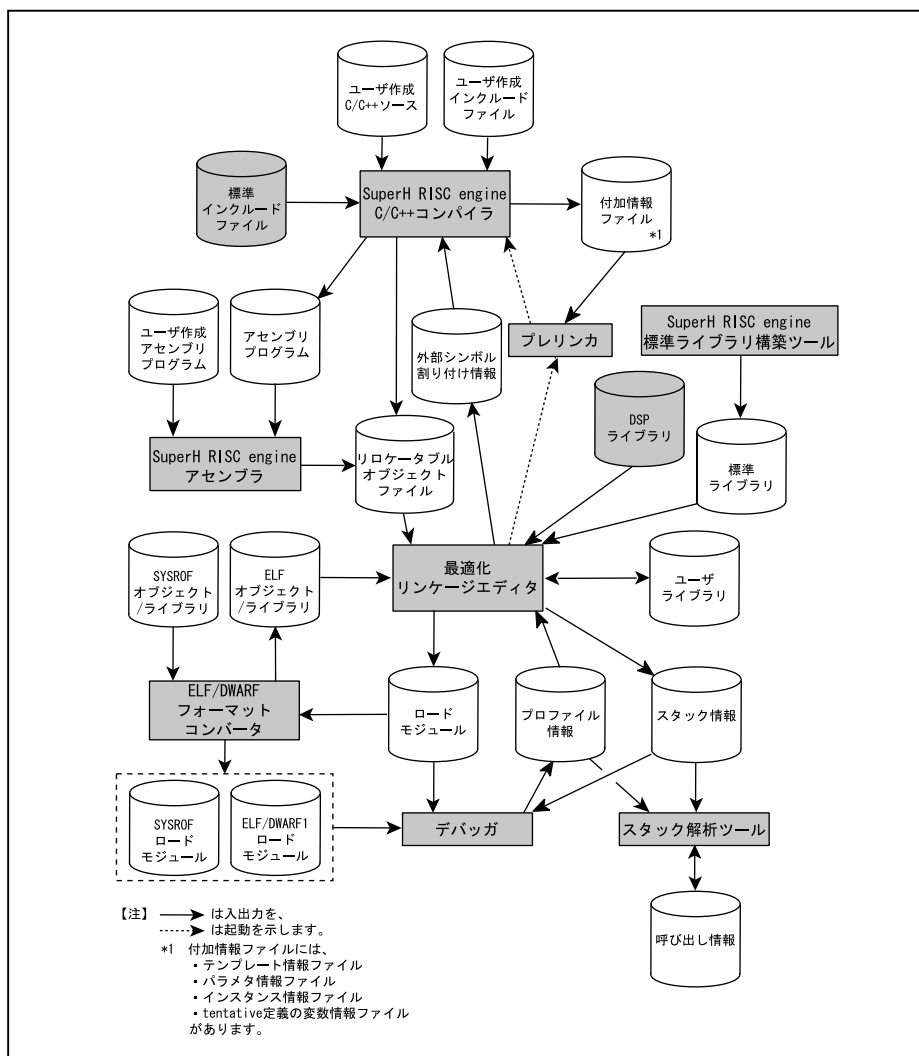


図 1.1 プログラムの開発手順

1. 概要

以下、C/C++コンパイラ、アセンブラ、最適化リンケージエディタ、プレリンカ、標準ライブラリ構築ツール、スタック解析ツール、フォーマットコンバータの概要を述べます。

1.2 コンパイラの概要

「SuperH RISC engine C/C++コンパイラ」(以下コンパイラと略します)は、C言語およびC++言語で記述したソースプログラムを入力し、SuperH RISC engine 用リロケータブルオブジェクトプログラムまたはアセンブリソースプログラムを出力します。

本コンパイラには次の特長があります。

- (1) 機器組み込み用としてROM化可能なオブジェクトプログラムを生成します。
- (2) オブジェクトプログラムの実行速度向上やサイズ縮小のための最適化機能をサポートしています。
- (3) プログラム記述言語として、C言語、C++言語をサポートしています。
- (4) C/C++言語でサポートしていない割り込み関数やシステム命令記述など、組み込み用プログラム作成に必要な機能を、拡張機能としてサポートしています。
- (5) デバッガによるC/C++ソースレベルデバッグを行うためのデバッグ情報出力を指定できます。
- (6) アセンブリソースプログラムまたはリロケータブルオブジェクトプログラムを選択して出力することができます。
- (7) 最適化リンケージエディタによるリンク時最適化を行うためのモジュール間最適化情報出力を指定できます。

1.3 アセンブラの概要

「SuperH RISC engine アセンブラ」(以下アセンブラと略します)は、アセンブリ言語で記述したソースプログラムを入力し、SuperH RISC engine 用リロケータブルオブジェクトプログラムを出力します。

本アセンブラには次の特長があります。

- (1) 次に示すプリプロセッサ機能により、効率よくソースプログラムを記述できます。
 - ファイルインクルード機能
 - 条件付アセンブリ機能
 - マクロ機能
- (2) 実行命令、アセンブラ制御命令のニーモニック(名称)は、IEEE-694仕様で規定された命名規則に準拠し、統一された体系となっています。

1.4 最適化リンケージエディタの概要

「最適化リンケージエディタ」は、コンパイラおよびアセンブラが出力した複数のオブジェクトプログラムを入力し、ロードモジュールまたはライブラリファイルを出力します。

本最適化リンケージエディタには次の特長があります。

- (1) コンパイラでは最適化できないメモリ配置や関数の呼び出し関係に依存した最適化をオブジェクトプログラムをまたがって実行します。
- (2) 以下の5種類のロードモジュールを選択出力できます。
 - リロケータブルELF形式
 - アブソリュートELF形式
 - Sタイプ形式
 - HEX形式
 - バイナリ形式
- (3) ライブラリファイルを作成・編集できます。

- (4) シンボル参照回数リストを出力できます。
- (5) ライブラリ、ロードモジュールファイルのデバッグ情報を削除できます。
- (6) スタック解析ツールで使用するスタック情報ファイルの出力を指定できます。

1.5 プレリンカの概要

「プレリンカ」は、最適化リンケージエディタから呼ばれ、C++プログラムのテンプレート、実行時型検査機能を使用している場合に、コンパイラを起動して必要なオブジェクトファイルを生成します。

通常はプレリンカを意識する必要はありませんが、C++プログラムのテンプレート、実行時型検査機能を使用していない場合、最適化リンケージエディタの `noprelink` オプションを指定してプレリンカの起動を抑止することにより、リンク速度を向上できます。

1.6 標準ライブラリ構築ツールの概要

「SuperH RISC engine 標準ライブラリ構築ツール」(以下標準ライブラリ構築ツールと略します)は、コンパイラが提供する標準ライブラリファイルをユーザ指定オプションで構築するソフトウェアシステムです。

本コンパイラが提供する標準ライブラリ関数には、C ライブラリ関数群、組み込み向け C++ クラスライブラリ関数群、実行時ルーチン群(プログラムを実行する上で必要な算術演算)があります。ソースプログラム上でライブラリ関数を使用しなくても、実行時ルーチンが必要な場合がありますので注意してください。

1.7 スタック解析ツールの概要

「スタック解析ツール」は、最適化リンケージエディタが出力したスタック情報ファイルを入力し、C/C++プログラムのスタック使用量を算出します。

1.8 フォーマットコンバータの概要

「ELF/DWARF フォーマットコンバータ」(以下フォーマットコンバータと略します)は、旧バージョンのコンパイラ、アセンブラ出力オブジェクトファイル、ライブラリファイルを入力し、ELF形式に変換します。または、アブソリュート ELF 形式のロードモジュールを入力し、旧バージョンのリンケージエディタ出力形式に変換します。

1. 概要

2. C/C++コンパイラ操作方法

2.1 オプション指定規則

コンパイラを起動するコマンドラインの形式は以下の通りです。

```
shc[ <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>][,...]
```

2.2 オプション解説

コマンドライン形式の場合は、英大文字は短縮形指定時の文字を、下線は省略時解釈を示します。
また、日立統合開発環境の対応するダイアログメニューを、タブ名[項目]で示します。
オプションの順序は、日立統合開発環境のタブに対応しています。

2.2.1 Source オプション

表 2.1 Source カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 インクルード ファイル ディレクトリ	Include = <パス名>[,...]	Source [Show entries for :] [Include file directories]	インクルードファイルの 取り込み先パス名を指定
2 デフォルト インクルード ファイル	PREInclude = <ファイル名>[,...]	Source [Show entries for :] [Preinclude]	指定したファイルをコンパイル 単位の先頭にインクルード
3 マクロ名の 定義	DEFine = <sub>[,...] <sub> : <マクロ名> [=<文字列>]	Source [Show entries for :] [Defines]	<文字列>を<マクロ名>として定義
4 インフォメー ション メッセージ	MESsage NOMESsage [=<エラー番号> [-<エラー番号>][,...]]	Source [Show entries for :] [Messages]	出力あり 出力なし

インクルードファイルディレクトリ

Include

	Source[Show entries for :][Include file directories]
書式	Include = <パス名>[,...]
説明	インクルードファイルの存在するパス名を指定します。 パス名が複数ある場合にはカンマ(,)で区切って指定することができます。 システムインクルードファイルの検索は、include オプション指定ディレクトリ、環境変数 SHC_INC 指定ディレクトリ、環境変数 SHC_LIB 指定ディレクトリの順序で行います。 ユーザインクルードファイルの検索は、カレントディレクトリ、include オプション指定ディレクトリ、環境変数 SHC_INC 指定ディレクトリ、環境変数 SHC_LIB 指定ディレクトリの順序で行います。
例	shc -include=c:¥usr¥inc,c:¥usr¥shc test.c ディレクトリ c:¥usr¥inc と c:¥usr¥shc をインクルードファイルパスとして検索します。

デフォルトインクルードファイル

PREInclude

	Source[Show entries for :][Preinclude]
書式	PREInclude = <ファイル名>[,...]
説明	指定したファイルの内容をコンパイル単位の先頭に取り込みます。ファイル名が複数ある場合にはカンマ(,)で区切って指定することができます。
例	shc -preinclude=a.h test.c <test.c>の内容 int a; main() { ... } コンパイル時解釈 #include "a.h" int a; main() { ... }

マクロ名定義

DEFine

Source[Show entries for :][Defines]

書 式 DEFine = <sub>[,...]
 <sub> : <マクロ名> [= <文字列>]

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

インフォメーションメッセージ

**MMessage
NOMessage**

Show[Show entries for :][Messages]

書 式 MMessage
 NOMessage [= <エラー番号> [- <エラー番号>][,...]]

説 明 インフォメーションレベルメッセージの出力有無を指定します。
 message オプションを指定した場合、インフォメーションレベルメッセージを出力します。
 nomessage オプションを指定した場合、インフォメーションレベルメッセージの出力を抑制
 します。また、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制します。<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。
 本オプションの省略時解釈は、nomessage です。

例 shc -message test.c
 インフォメーションレベルメッセージの出力を指定します。

備 考 C5000 以降のインフォメーションメッセージは、出力を抑制することができません。

2.2.2 Object オプション

表 2.2 Object カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 プリプロセッサ展開	PREProcessor [=<ファイル名>]	Object [Output file type :] [Preprocessed source file]	プリプロセッサ展開後のソースプログラムを出力
2 オブジェクト形式	Code = { Machinecode Asmcode }	Object [Output file type :] [Machine code] [Assembly source code]	機械語プログラムを出力 アセンブリプログラムを出力
3 デバッグ情報	DEBug NODEBug	Object [Generate debug information]	出力あり 出力なし
4 セクション名	SEction = <sub>[,...] <sub>:{ Program=<セクション名> Const = <セクション名> Data = <セクション名> Bss = <セクション名> }	Object [section :] [Program section (P)] [Const section (C)] [Data section (D)] [Uninitialized data section (B)]	プログラム領域のセクション名 定数領域のセクション名 初期化データ領域のセクション名 未初期化データ領域のセクション名
5 文字列出力領域	SString = { Const Data }	Object [Store string data in :]	定数領域へ出力 初期化データ領域へ出力
6 オブジェクトファイル名指定	OBjectfile = <ファイル名>	Object [Output directory :]	指定したファイル名のオブジェクトファイルを出力
7 テンプレートインスタンス生成機能	Template = { None Static Used ALI AUto }	Object [Template :]	インスタンスを生成しません 参照されたものだけ内部リンケージとして生成します 参照されたものだけ外部リンケージとして生成します 宣言、参照されたものを生成します リンク時に生成します
8 ABS16 宣言	ABs16 = { RU ALI }	Object [Use 16 bit short address]	実行時ルーチンをすべて #pragma abs16 宣言されたものとみなします。 すべてのラベルアドレスを 16 ビットで生成します。
9 除算方式選択 (SH-2 ~ SH-4)	Division = Cpu = { Inline Runtime }	Object [Division sub-options:]	CPU の除算命令を使用 乗算に変換しインライン展開 実行時ルーチン呼び出し

プリプロセッサ展開

PREProcessor

	Object[Output file type :][Preprocessed source file]
書 式	PREProcessor [= <ファイル名>]
説 明	<p>プリプロセッサ展開後のソースプログラムを出力します。</p> <p><ファイル名>を指定しない場合は、ソースファイル名と同じファイル名で拡張子が「p」（入力ソースファイルがCプログラムの場合）、または「pp」（入力ソースプログラムがC++プログラムの場合）のファイルが作成されます。</p> <p>本オプションを指定した場合は、オブジェクトプログラムを出力しません。</p>
備 考	<p>本オプションを指定した場合、以下のオプションが無効になります。</p> <p>code、debug、section、string、objectfile、template、abs16、 show=object、statistics、optimize、speed、nospeed、size、goptimize、inline、 case、macsave、align16、rtnext、loop、fpscr、cpu、division、endian、fpu、 round、denormalize、pic、double=float、exception、rtti、outcode、map、gbr、 shift、blockcopy、approxdiv、patch=7055</p>

オブジェクト形式

Code

	Object[Output file type :][Machine code][Assembly source code]
書 式	Code = { <u>Machinecode</u> Asmcode }
説 明	<p>オブジェクトプログラムの出力形式を指定します。</p> <p>code=machinecode を指定した場合、リロケータブルオブジェクト(機械語)プログラムを出力します。</p> <p>code=asmcode を指定した場合、アセンブリプログラムを出力します。</p> <p>本オプションの省略時解釈は、code=machinecode です。</p>
備 考	code=asmcode を指定した場合、show=object、goptimize オプションは無効になります。

**DEBug
NODEBug**

Object[Generate debug information]

書式	DEBug NODEBug
説明	debug オプションを指定した場合、ソースレベルデバッグに必要なデバッグ情報をオブジェクトファイルに出力します。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 オプションを指定した場合、デバッグ情報をオブジェクトファイルに出力しません。本オプションの省略時解釈は、nodebug です。

SEctionObject[Section :][Program section (P)][Const section (C)]
[Data section (D)][Uninitialized data section (B)]

書式	SEction = <sub>[,...] <sub>: { Program = <セクション名> Const = <セクション名> Data = <セクション名> Bss = <セクション名> }
説明	オブジェクトプログラム中のセクション名を指定します。 section=program=<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=const=<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=data=<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=bss=<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。 <セクション名>は、英字、数字、アンダーライン(_)または、\$の列で、先頭が数字以外のものです。セクション名は、8192文字目まで有効です。 本オプションの省略時解釈は、section=program=P,const=C,data=D,bss=B です。
備考	プログラムとセクション名の対応についての詳細は、「9.1 プログラムの構造」を参照してください。

文字列出力領域

String

Object[Store string data in :]

書 式 String = { Const | Data }

説 明 文字列の出力先を指定します。
 string=const を指定した場合、定数領域に出力します。
 string=data を指定した場合、初期化データ領域に出力します。
 初期化データ領域へ出力した文字列はプログラム実行時に変更することができますが、ROM
 上と RAM 上に二重に領域を確保し、プログラム実行開始時に ROM から RAM へ転送する必要が
 あります。初期化データ領域の初期設定、メモリ割り付けの方法については、「9.2.1 メモ
 リ領域の割り付け」を参照してください。
 本オプションの省略時解釈は、string=const です。

オブジェクトファイル名指定

Objectfile

Object[Output directory :]

書 式 Objectfile = <ファイル名>

説 明 出力するオブジェクトファイル名を指定します。
 本オプション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「obj」(出
 力ファイルがリロケータブルオブジェクトプログラムの場合)、または「src」(出力ファイ
 ルがアセンブリソースプログラムの場合)のオブジェクトファイルを出力します。
 ファイル拡張子が「obj」か「src」かは、code オプションで決まります。

テンプレートインスタンス生成機能

Template

		Objct[Template :]
書 式	<pre> Template = { None Static Used ALl AUto } </pre>	
説 明	<p>テンプレートのインスタンス生成方法を指定します。</p> <p>template=none を指定した場合、インスタンスの生成を行いません。</p> <p>template=static を指定した場合、コンパイル単位内で参照されたテンプレートのみインスタンスを作成します。ただし、生成される関数は内部リンケージを持ちます。</p> <p>template=used を指定した場合、コンパイル単位内で参照されたテンプレートのみインスタンスを作成します。ただし、生成される関数は外部リンケージを持ちます。</p> <p>template=all を指定した場合、コンパイル単位内で宣言または参照されている全てのテンプレートのインスタンスを作成します。</p> <p>template=auto を指定した場合、リンク時に必要なインスタンスの生成を行います。</p>	
備 考	code=asmcode を指定した場合は、常に template=static になります。	

ABS16 宣言**ABS16**

		Objct[Use 16 bit short address]
書 式	<pre> ABS16 = { RUn ALl } </pre>	
説 明	<p>abs16=run を指定した場合、すべての実行時ルーチンが #pragma abs16 宣言されたものとみなします。</p> <p>abs16=all を指定した場合、すべてのラベルアドレスを 16 ビットで生成します。</p>	

除算方式選択

***D*ivision**

```

Object[Division sub-options:]/CPU[Division:]
書 式  DDivision = { Cpu [= { Inline | Runtime }] |
                Peripheral |
                Nomask }

```

- 説明 プログラム中の整数型除算、剰余算の方式を選択します。
- division=cpu=inline を指定した場合、定数除算は乗算に変換してインライン展開し、変数除算は DIV1 命令による実行時ルーチンを選択します。
- division=cpu=runtime を指定した場合、シフト演算で行えない除算について DIV1 命令による実行時ルーチンを選択します。
- division=cpu のみ指定した場合は、size オプション指定時は division=cpu=runtime、speed、nospeed オプション指定時は division=cpu=inline とみなします。
- division=peripheral を指定した場合、除算器を用いた実行時ルーチンを選択(割り込みマスクに 15 を設定)します。CPU が SH-2(SH7604)の時のみ実行可能です。
- division=nomask を指定した場合、除算器を用いた実行時ルーチンを選択(割り込みマスクは変更なし)します。CPU が SH-2(SH7604)の時のみ実行可能です。
- peripheral、nomask 指定時は以下の点に注意してください。
- (1) ゼロ除算のチェックおよび errno の設定は行いません。
 - (2) nomask 指定時には、除算器動作中に割り込みがかかり、割り込み処理ルーチンで除算器を用いた場合、動作は保証しません。
 - (3) オーバフロー割り込みはサポートしていません。
 - (4) ゼロ除算、オーバフローなどの演算結果は除算器の仕様に従います。cpu サブオプション指定時と異なる場合があります。
- 本オプションは、cpu=sh1 を指定した場合は無効になります。

2.2.3 List オプション

表 2.3 List カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 リスト ファイル	Listfile [= <ファイル名>] NOListfile	List [Generate list file]	出力あり 出力なし
2 リスト内容と 形式	SHow = <sub>{,...} <sub>:{ SOurce NOSource ObjeCt NOObject SStatistics NOSTatistics Include NOInclude Expansion NOExpansion Width = <数値> Length = <数値> Tab = {4 8} }	List [Contents]	ソースリストの有無 オブジェクトリストの有無 統計情報の有無 インクルード展開後リストの有無 マクロ展開後リストの有無 1行の最大文字数: 0, 80 ~ 132 ページ内の最大行数: 0, 40 ~ 255 タブ使用時のカラム数: 4 か 8

リストファイル

Listfile NOListfile

List[Generate list file]

書 式 Listfile [= <ファイル名>]
 NOListfile

説 明 リストファイルの出力有無を指定します。
listfile オプションを指定した場合、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。
nolistfile オプションを指定した場合、リストファイルは出力しません。
<ファイル名>は、「8.1 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<リストファイル名>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 UNIX 版のとき「lis」、PC 版のとき「lst」（入力ソースが C プログラムの場合）、または「lpp」（入力ソースが C++ プログラムの場合）のリストファイルが作成されます。
本オプションの省略時解釈は、nolistfile です。

リスト内容と形式

***S*Show**

List[Contents]

```
書 式  SHow = <sub>[,...]
        <sub>:{ SSource      |NOSource      |
              Object      |NOObject      |
              SStatistics  |NOSTatistics  |
              Include     |NOInclude     |
              Expansion    |NOExpansion   |
              Width = <数値> |
              Length = <数値> |
              Tab = { 4 | 8 } }
```

説 明 コンパイラが出力するリストの内容とその形式、および出力の解除を指定します。
本項で記した各リストの具体例については「8.2 コンパイルリストの参照方法」を参照してください。
本オプションの省略時解釈は、show=nosource,object,statistics,noinclude, noexpansion,width=0,length=0,tab=8 です。

備 考 サブオプションの一覧を表 2.4 に示します。

表 2.4 show オプションのサブオプション一覧

サブオプション名	意味
1 source	ソースプログラムのリストを出力します。
2 nosource	ソースプログラムのリストを出力しません。
3 object	オブジェクトプログラムのリストを出力します。
4 noobject	オブジェクトプログラムのリストを出力しません。
5 statistics	統計情報のリストを出力します。
6 nostatistics	統計情報のリストを出力しません。
7 include	インクルードファイル展開した後のソースプログラムリストを出力します。 nosource サブオプションが同時に指定された場合には、include サブオプションは無効となり、ソースプログラムリストは出力されません。
8 noinclude	インクルードファイル展開する前のソースプログラムリストを出力します。 nosource サブオプションが同時に指定された場合には、noinclude サブオプションは無効となり、ソースプログラムリストは出力されません。
9 expansion	マクロ展開した後のソースプログラムリストを出力します。nosource サブオプションが同時に指定された場合には、expansion サブオプションは無効となり、ソースプログラムリストは出力されません。
10 noexpansion	マクロを展開する前のソースプログラムリストを出力します。nosource サブオプションが同時に指定された場合には、noexpansion サブオプションは無効となり、ソースプログラムリストは出力されません。
11 width = <数値>	<数値>で指定する文字数をリストの 1 行の最大文字数とします。<数値>は 10 進数で指定し、0 または 80 から 132 の間の数値を指定することができます。 <数値>が 0 の場合、リストの 1 行の最大文字数は規定されません。
12 length = <数値>	<数値>で指定する行数を、リストの 1 ページの最大行数とします。<数値>は 10 進数で指定し、0 または 40 から 255 の間の数値を指定することができます。 <数値>が 0 の場合、リストの 1 ページの最大行数は規定されません。
13 tab = { 4 8 }	リスト表示時のタブのサイズを指定します。

2.2.4 Optimize オプション

表 2.5 Optimize カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 最適化レベル	OPTimize = { 0 1 }	Optimize [Optimization]	最適化なし 最適化あり
2 スピード・サイズ最適化	SPeed Slze NOSPeed	Optimize [Speed or size :] [Optimize for speed] [Optimize for size] [Optimize for both speed and size]	最適化項目の選択
3 モジュール間最適化	Goptimize	Optimize [Generate file for inter-module optimization]	モジュール間最適化用付加情報出力
4 外部変数アクセス最適化	MAP = <ファイル名>	Optimize [Include map file :]	外部変数アクセス最適化を行う
5 GBR 相対アクセスコード自動生成	GBr = { Auto User }	Optimize [Gbr relative logic operation :]	GBR 相対アクセスコード自動生成を行う GBR 相対アクセスコード自動生成を行わない
6 switch 文展開方式	CAsE = { Ifthen Table }	Optimize [Switch statement :]	if_then 方式で展開 テーブルジャンプ方式で展開
7 シフト演算展開	SHift = { Inline Runtime }	Optimize [Shift operation :]	インライン展開する 展開命令数の多いシフト演算を実行時ルーチン呼び出しにする
8 転送コード展開	BLOckcopy = { Inline Runtime }	Optimize [Transfer code development :]	インライン展開する 転送サイズが大きい場合は実行時ルーチン呼び出しにする
9 自動インライン展開	INLine[= <数値>] NOINLine	Optimize [Automatic inline expansion]	自動インライン展開する 自動インライン展開しない

最適化レベル

Optimize

Optimize[Optimization]

書 式 OPTimize = { 0 | 1 }

説 明 オブジェクトプログラムの最適化レベルを指定します。
optimize=0 を指定した場合、オブジェクトプログラムの最適化を行いません。
optimize=1 を指定した場合、最適化を行います。
本オプションの省略時解釈は、optimize=1 です。

スピード・サイズ最適化

SPeed
SIZe
NOSPeed

Optimize[Speed or size :][Optimize for speed][Optimize for size]
[Optimize for both speed and size]

書 式 *SPeed*
 SIZe
 NOSPeed

説 明 *speed*、*size*、*nospeed* オプションにおける主な最適化項目を表 2.6 に示します。
 これらの最適化項目は、オプションで制御可能です。
 本オプションの省略時解釈は、*nospeed* です。

表 2.6 最適化項目一覧

	自動インライン展開	ループ展開最適化	シフトコード展開	転送コード展開	整数定数除算展開
<i>speed</i>			<i>inline</i>	<i>inline</i>	<i>inline</i>
<i>size</i>	x	x	<i>runtime</i>	<i>runtime</i>	<i>runtime</i>
<i>nospeed</i>	x	x	<i>inline</i>	<i>inline</i>	<i>inline</i>

モジュール間最適化

Goptimize

Optimize[Generate file for inter-module optimization]

書 式 *Goptimize*

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。

MAP

Optimize[Include map file :]

書 式 MAP = <ファイル名>

説 明 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。gbr=autoを指定した場合、条件によりベースアドレスを GBR レジスタに設定し、外部変数アクセスを GBR 相対で行うコードを生成します。
本オプションを使用する際は、プログラムを本オプションを指定しないで一度コンパイルし、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 map=<ファイル名>を指定してコンパイルしてください。

例 プログラム(test.c)

```
int A,B,C;
void main()
{
    A = 0;
    B = 0;
    C = 0;
}
```

(1) コマンド: shc test.c

<出力コード>

```
_main:
    MOV.L    L11,R6    ;_A
    MOV     #0,R2
    MOV.L    R2,@R6
    MOV.L    L11+4,R6  ;_B
    MOV.L    R2,@R6
    MOV.L    L11+8,R6 ;_C
    RTS
    MOV.L    R2,@R6
L11:
    .DATA.L  _A
    .DATA.L  _B
    .DATA.L  _C
```

(2) コマンド: shc test.c
optlnk -map=test.bls -start=P/400,B/1000 test.obj
shc -map=test.bls test.c

リンク後のデータ配置

A	0x1000
B	0x1004
C	0x1008

<出力コード>

```
_main:
    MOV.W    L11,R1    ;_A      Aのアドレスをベースアドレスに設定
    MOV      #0,R0
    MOV.L    R0,@R1
    MOV.L    R0,@(4,R1)
    RTS
    MOV.L    R0,@(8,R1)

L11:
    .DATA.W  _A        Aのアドレスは2バイト
```

- 備考 外部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。
- 本オプション以外で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、動作は保証されません。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なおしてください。

GBr

Optimize[Gbr relative logic operation :]

書 式 GBr = { Auto | User }

説 明 gbr=auto を指定した場合、条件によりコンパイラが自動的に GBR 相対論理演算コードを生成します。gbr=auto かつ map=<ファイル名>を指定した場合は、条件により GBR にベースアドレスを設定し、外部変数アクセスを GBR 相対で行うコードを生成します。
gbr=user を指定した場合、GBR の設定、参照、GBR 相対アクセスは #pragma gbr_base、#pragma gbr_base1、または GBR 関連組み込み関数でユーザが指定します。
本オプション省略時解釈は、gbr=auto です。

例 プログラム

```
char A,B,C;
void main()
{
    A |= 1;
    B &= 1;
    C ^= 1;
}
```

<出力コード(gbr=auto)>

```
_main:
    STC        GBR,@-R15    ; GBR を退避
    MOV        #0,R0
    LDC        R0,GBR      ; GBR に 0 を設定
    MOV.L     L11+2,R0     ; R0 <- A のアドレス
    OR.B      #1,@(R0,GBR) ; A |= 1
    MOV.L     L11+6,R0     ; R0 <- B のアドレス
    AND.B     #1,@(R0,GBR) ; B &= 1
    MOV.L     L11+10,R0    ; R0 <- C のアドレス
    XOR.B     #1,@(R0,GBR) ; C ^= 1
    RTS
    LDC        @R15+,GBR   ; GBR の回復
L11:
    .RES.W    1
    .DATA.L   _A
    .DATA.L   _B
    .DATA.L   _C
```

備 考 #pragma gbr_base、#pragma gbr_base1 を使用したプログラムを gbr=auto を指定してコンパイルするとウォーニングメッセージを出力し、#pragma 指定を無視します。また、gbr 関連組み込み関数を使用したプログラムを gbr=auto を指定してコンパイルするとエラーになります。
gbr=auto を指定した場合、関数の呼び出し前後で GBR レジスタを保証します。

switch 文展開方式***C*Ase**

Optimize[Switch statement :]

書 式 C*A*se = { Ifthen | Table }

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* ラベルの飛び先をジャンプテーブルに確保し、1回のジャンプテーブルの参照で *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定数領域に確保されるジャンプ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。

本オプション省略時は、いずれかの展開方式をコンパイラが自動的に選択します。

シフト演算展開

***S*Hift**

Optimize[Shift Operation :]

書 式 *S*Hift = { Inline | Runtime }

説 明 *shift=inline* を指定した場合、すべてのシフト演算を命令展開します。

shift=runtime を指定した場合、展開する命令が多い場合は実行時ルーチン呼び出しになります。

size オプションを指定した場合のデフォルトは *shift=runtime* です。また *speed*、*nospeed* オプションを指定した場合のデフォルトは *shift=inline* です。

BLOckcopy

Optimize [Transfer code development :]

書 式 BLOckcopy = { Inline | Runtime }

説 明 blockcopy=inline を指定した場合、すべてのメモリ間転送コードを命令展開します。
blockcopy=runtime を指定した場合、転送サイズが大きい場合は実行時ルーチン呼び出しになります。
size オプションを指定した場合のデフォルトは、blockcopy=runtime です。また speed、nospeed オプションを指定した場合のデフォルトは、blockcopy=inline です。

INLine
NOINLine

Optimize[Automatic inline expansion]

書 式 INLine [= <数値>]
NOINLine

説 明 関数の自動インライン展開を行うかどうかを指定します。
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。
inline=<数値>で、プログラムサイズが何%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=50 を指定した場合、プログラムサイズが 50%増加するまで(1.5倍になるまで)インライン展開を行います。
noinline 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せん。
Speed オプションを指定した場合のデフォルトは、inline=20 です。nospeed、size オプションまたは optimize=0 を指定した場合のデフォルトは、noinline です。

2.2.5 Other オプション

表 2.7 Other カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 組み込み向け C++言語	ECpp	Other [Miscellaneous options :] [Check against EC++ language specification]	EC++言語仕様に基づいてシンタックスチェック
2 コメントの ネスト	COMment = { Nest NONest }	Other [Miscellaneous options :] [Allow comment nest]	コメント(/** */)のネストを許す コメント(/** */)のネストを許さない
3 MAC レジスタ保証	Macsave = { 0 1 }	Other [Miscellaneous options :] [Callee saves/restores MACH and MACL registers if used]	関数呼び出し前後で MAC レジスタを保証しない 関数呼び出し前後で MAC レジスタを保証する
4 ラベルの 16 バイト整合	ALign16 NOALign16	Other [Miscellaneous options :] [Align labels after unconditional branches 16byte boundaries]	プログラムセクション内のラベルで、サブルーチンコール以外の無条件分岐命令直後のラベルをすべて 16 バイト整合する。 ラベルを 16 バイト整合しない。
5 リターン値の 拡張	RTnext NORTnext	Other [Miscellaneous options :] [Expand return value to 4 byte]	返却値の符号/ゼロ拡張する 返却値の符号/ゼロ拡張しない
6 ループ展開最 適化	LOop NOLOop	Other [Miscellaneous options :] [Loop unrolling]	ループ展開の最適化を行う ループ展開の最適化を行わない
7 浮動小数点定 数除算の乗算 化	AApproxdiv	Other [Miscellaneous options :] [approximate a floating-point constant division]	浮動小数点定数除算の乗算化を行う
8 SH7055 不具 合回避	PArch=7055	Other [Miscellaneous options :] [Avoid illegal SH7055 instructions]	SH7055 で不正動作となる命令並びを生成しない
9 FPSCR レジ スタの切り替 え (SH-4 のみ)	FPScr = { Safe Aggressive }	Other [Miscellaneous options :] [Change FPSCR register if double data used]	FPSCR レジスタを double 演算が発生するごとに切り替え FPSCR レジスタの切り替えを可能な限り抑止
10 ループ判定式 の最適化抑止	Volatile_loop	Other [Miscellaneous options :] [Treats loop condition as volatile qualified]	ループ判定式の最適化を抑止する

ECpp

	<code>Other[Miscellaneous options :][Check against EC++ language specification]</code>
書 式	<code>ECpp</code>
説 明	Embedded C++言語仕様に基づいて、C++プログラムのシンタックスチェックを行います。Embedded C++言語仕様では、 <code>catch</code> 、 <code>const_cast</code> 、 <code>dynamic_cast</code> 、 <code>explicit</code> 、 <code>mutable</code> 、 <code>namespace</code> 、 <code>reinterpret_cast</code> 、 <code>static_cast</code> 、 <code>template</code> 、 <code>throw</code> 、 <code>try</code> 、 <code>typeid</code> 、 <code>typename</code> 、 <code>using</code> をサポートしていません。これらのキーワードを記述した場合、エラーメッセージを出力します。
備 考	Embedded C++言語仕様では、多重継承、仮想基底クラスをサポートしていません。多重継承、仮想基底クラスを記述した場合は、ウォーニングメッセージ "C5882 (W) Embedded C++ does not support multiple or virtual inheritance" を出力します。ウォーニングメッセージ C5882 出力時のコンパイラ生成オブジェクトプログラムは、本オプションを指定しない場合と変わりません。

COMment

	<code>Other[Miscellaneous options :][Allow comment nest]</code>
書 式	<code>COMment = { Nest NONest }</code>
説 明	<code>comment=nest</code> 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。 <code>comment=nonest</code> を指定した場合、コメントのネストを記述するとエラーになります。 本オプションの省略時解釈は、 <code>comment=nonest</code> です。
例	<code>/* This is an example of /* nested */ comment */</code>
	[1] <code>comment=nest</code> を指定した場合は全てコメントと解釈しますが、 <code>comment=nonest</code> 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。

MAC レジスタ保証**Macsave**

	Other[Miscellaneous options :]
	[Callee saves/restores MACH and MACL registers if used]
書 式	Macsave = { 0 1 }
説 明	<p>MACH、MACL レジスタを関数の呼び出し前後で保証するかどうかを指定します。</p> <p>macsave=0 を指定した場合、関数の呼び出し前後で MACH、MACL レジスタを保証しません。</p> <p>macsave=1 を指定した場合、関数の呼び出し前後で MACH、MACL レジスタを保証します。</p> <p>macsave=1 でコンパイルした関数から macsave=0 でコンパイルした関数を呼び出すことはできません。逆に macsave=0 でコンパイルした関数から macsave=1 でコンパイルした関数を呼び出すことは可能です。</p> <p>本オプションの省略時解釈は、macsave=1 です。</p>

ラベルの 16 バイト整合**ALign16
NOALign16**

	Other[Miscellaneous options :]
	[Align Labels after unconditional branches 16byte boundaries]
書 式	ALign16 NOALign16
説 明	<p>align16 オプションを指定した場合、プログラムセクション内のラベルでサブルーチンコール以外の無条件分岐命令直後のラベルを、すべて 16 バイト整合します。</p> <p>noalign16 オプションを指定した場合、サブルーチンコール以外の無条件分岐命令直後のラベルを 16 バイト整合しません。</p> <p>本オプションの省略時解釈は、noalign16 です。</p>

RTnext
NORNext

Other[Miscellaneous options :][Expand return value to 4 byte]

書 式 RTnext
NORNext

説 明 (unsigned) char 型、(unsigned) short 型を返す return 文において、関数返却値レジスタ R0 の符号拡張あるいは、ゼロ拡張を行うかどうか指定します。関数プロトタイプ宣言がある場合は、本オプションを指定する必要はありません。

rtnext オプションを指定した場合、関数返却値の符号/ゼロ拡張を行います。

nortnext オプションを指定した場合、関数返却値の符号/ゼロ拡張を行いません。

本オプションの省略時解釈は、nortnext です。

LOop
NOLoop

Other[Miscellaneous options :][Loop unrolling]

書 式 LOop
NOLoop

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。

noloop オプションを指定した場合、ループ文を展開しません。

本オプションの省略時解釈は、noloop です。

浮動小数点定数除算の乗算化***A*pproxdiv**

```
Other[Miscellaneous options :][approximate a floating-point constant division]
```

書 式 `A`pproxdiv

説 明 浮動小数点定数除算を、定数の逆数の乗算に変換します。

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、演算の精度が変わる場合がありますので注意が必要です。

SH7055* 不具合回避**P*Atch**

```
Other[Miscellaneous options :][Avoid illegal SH7055 instructions]
```

書 式 `P`Atch = 7055

説 明 SH7055 で不正動作となる命令並びをコンパイラで出力しないようにします。

備 考 本オプションは、`cpu=sh2e` を指定した場合のみ有効です。

FPSCR レジスタ精度モード切り替え

FPScr

Other[Miscellaneous options :][Change FPSCR register if double data used]
書 式 FPScr = { Safe | Aggressive }

説 明 FPSCR レジスタの精度モードを関数呼び出し前後で保証するかどうか指定します。
SH-4 では float 演算、double 演算を実行するときに FPSCR レジスタの精度モードを単精度、倍精度に設定します。
fpscr=safe を指定した場合、double 演算が出現するたびに FPSCR レジスタの精度モードを倍精度モードに切り替えます。倍精度演算が終了すると単精度モードに切り替えます。この場合、関数呼び出しから戻ったときの FPSCR レジスタの精度モードは常に単精度モードになります。
fpscr=aggressive を指定した場合、double 演算が続く場合は可能な限り FPSCR レジスタの精度モードの切り替えが起こらないようにします。この場合、関数呼び出しから戻ったときの FPSCR レジスタの精度モードの値は保証されません。
本オプションは、cpu=sh4 で fpu=single、fpu=double のどちらの指定もないときに有効です。
本オプションの省略時解釈は、fpscr=aggressive です。

ループ判定式最適化抑止

Volatile_loop

Other[Miscellaneous options :][Treats loop condition as volatile qualified]
書 式 Volatile_loop

説 明 ループ判定式に外部変数を含む場合、最適化対象外にします。
ただし、型変換を伴う場合、外部変数を 2 つ以上含む場合または複合演算の場合は、最適化抑止対象にならない場合があります。

備 考 本オプションを指定しなかった場合、ループ判定式がループ内で不変の時に削除される場合があります。

2.2.6 CPU オプション

表 2.8 CPU タブオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 CPU / 動作モード	CPU = { SH1 SH2 SH2E SH3 SH4 }	CPU [CPU :]	SH-1 のオブジェクトを生成 SH-2 のオブジェクトを生成 SH-2E のオブジェクトを生成 SH-3 のオブジェクトを生成 SH-4 のオブジェクトを生成
2 メモリのバイト並び順 (SH-3 ~ SH-4)	ENdian = { Big Little }	CPU [Endian :]	Big Endian Little Endian
3 浮動小数点演算モード (SH-4 のみ)	FPU = { Single Double }	CPU [FPU :]	浮動小数点演算をすべて単精度浮動小数点で演算 浮動小数点演算をすべて倍精度浮動小数点で演算
4 丸め方式 (SH-4 のみ)	Round = { Zero Nearest }	CPU [Round to :]	Round to Zero で丸める Round to Nearest で丸める
5 非正規化数の扱い (SH-4 のみ)	DENormalize = { OFF ON }	CPU [Denormalized number allow as a result]	非正規化数を 0 として扱う 非正規化数を非正規化数として扱う
6 プログラムセクションインディペンデント (SH-2 ~ SH-4)	Pic = { 0 1 }	CPU [Position independent code (PIC)]	プログラムセクションのポジションインディペンデントコードを生成しない プログラムセクションのポジションインディペンデントコードを生成する
7 double float 変換 (SH-1 ~ SH-3)	DOuble = Float	CPU [Treat double as float]	double 型の変数を float 型として扱う
8 例外処理機能	EXception NOEXception	CPU [Use try,throw and catch of C++]	例外処理機能を有効にする 例外処理機能を無効にする
9 実行時型情報	RTTI = { ON OFF }	CPU [Enable/disable runtime information]	dynamic_cast, typeid を有効にする dynamic_cast, typeid を無効にする
10* 除算方式選択 (SH-2 のみ)	Dlvision = { Cpu Peripheral Nomask }	CPU [Division :]	CPU の除算命令を使用 除算器を使用 (割り込みマスクあり) 除算器を使用 (割り込みマスクなし)

【注】* オプションの詳細は「2.2.2 オブジェクトオプション」を参照してください。

CPu

CPU[CPU :]

書 式 CPu = { SH1 |
 SH2 |
 SH2E |
 SH3 |
 SH4 }

説 明 作成するオブジェクトプログラムの CPU 種別を指定します。サブオプションの一覧を
 表 2.9 に示します。
 本オプションの省略時解釈は、cpu=sh1 です。

表 2.9 cpu オプションのサブオプション一覧

サブオプション名	意 味
1 sh1	SH-1 のオブジェクトを作成します。
2 sh2	SH-2 のオブジェクトを作成します。
3 sh2e	SH-2E のオブジェクトを作成します。
4 sh3	SH-3 のオブジェクトを作成します。
5 sh4	SH-4 のオブジェクトを作成します。

ENdian

CPU[Endian :]

書 式 ENdian = { Big | Little }

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。
 Little Endian のオブジェクトプログラムは、SH-1、SH-2、SH-2E では実行できません。
 本オプションの省略時解釈は、endian=big です。

浮動小数点演算モード

FPU

CPU[FPU :]

書 式 FPU = { Single | Double }

説 明 fpu=single を指定した場合、すべての浮動小数点演算を単精度浮動小数点で演算します。
fpu=double を指定した場合、すべての浮動小数点演算を倍精度浮動小数点で演算します。
プログラム中に浮動小数点演算がない場合には、fpu=single を指定してください。
本オプションは、cpu=sh4 を指定した場合のみ有効です。

丸め方式

Round

CPU[Round to :]

書 式 Round = { Zero | Nearest }

説 明 浮動小数点定数をオブジェクトコードに変換する際の丸め方式を指定します。
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。
本オプションは、cpu=sh4 を指定した場合のみ有効です。
本オプションの省略時解釈は、round=zero です。

DENormalize

CPU[Denormalized number allowed as a result]

書 式 DENormalize = { OFF | ON }

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。
本オプションは、cpu=sh4 を指定した場合のみ有効です。
本オプションの省略時解釈は、denormalize=off です。

ポジションインディペンデントコード***Pic***

CPU[Position independent code (PIC)]

書 式 Pic = { 0 | 1 }

説 明 pic=1 を指定した場合、リンク後のプログラムセクションを任意のアドレスに配置して実行できます。データセクションはリンク時に決定したアドレス以外には配置できません。ポジションインディペンデントコードとして実行する場合は、関数のアドレスを初期値として指定することはできません。C++コンパイルでは、仮想関数、関数メンバへのポインタも関数のアドレスを初期値として必要とするため、仮想関数やメンバ関数へのポインタを含んだC++プログラムは、ポジションインディペンデントコードとして実行できません。

例 1

```
extern int f();
int (*fp)() = f;    指定不可
```

例 2

```
struct A{virtual void f();};    指定不可
void (A::*ap)() = &A::f;    指定不可
```

cpu=sh1 を指定した場合は、pic=1 指定を無視します。

本オプションの省略時解釈は、pic=0 です。

*double float 変換****D*Ouble = *F*loat**

CPU[Treat double as float]

書式	D <code>O</code> uble = <code>F</code> loat
説明	double(倍精度浮動小数点)型の数値を float(単精度浮動小数点)型としてオブジェクトを生成します。
備考	cpu=sh4 を指定した場合、本オプションを指定すると本オプションは無効となり、fpu=single が指定されたとみなします。

例外処理機能***EX*ception
*NOEX*ception**

CPU[Use try,throw and catch of C++]

書式	<code>EX</code> ception <code>NOEX</code> ception
説明	exception オプションを指定した場合、C++例外処理機能(try,catch,throw)を有効にします。 noexception オプションを指定した場合、C++例外処理機能(try,catch,throw)を無効にします。 exception オプションを指定した場合、コード性能が低下する可能性があります。 本オプション省略時解釈は、noexception です。
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。 (1) rtti=on を指定する。 (2) 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。

RTTI

CPU[Enable/disable runtime information]

書 式 RTTI = { ON | OFF }

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。
 本オプション省略時解釈は、rtti=off です。

備 考 本オプションを指定して作成したオブジェクトファイルをライブラリに登録したり、リロケータブルオブジェクトファイルに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。

2.2.7 その他オプション

表 2.10 その他のオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 C/C++言語の 選択	LAng = { C C++ } C++ }	- (拡張子で判断)	C プログラムとしてコンパイル C++プログラムとしてコンパイル
2 コピーライト 出力抑止	LOGO NOLOGO	- (常に nologo が有効)	コピーライトを出力 コピーライトの出力を抑止
3 文字列内の 文字コード	Euc SJis LATin1	-	euc コードを選択 sjis コードを選択 latin1 コードを選択
4 オブジェクト 内漢字コード 指定	OUnicode = { EUc SJis }	-	euc コード sjis コード
5 サブコマンド ファイルの 指定	SUBcommand = <ファイル名>	-	<ファイル名>で指定したファイル からコマンドオプションを取りこ む

C/C++言語選択

LAng

なし (常に拡張子で判断)

書 式 LAng = { C | C++ }

説 明 ソースプログラムの言語を指定します。
 lang=c を指定した場合、C プログラムとしてコンパイルします。
 lang=c++ を指定した場合、C++ プログラムとしてコンパイルします。
 本オプションを省略した場合は、ソースプログラムの拡張子によって判断します。拡張子が c
 のときには C プログラムとしてコンパイルします。また、拡張子が cpp、cc、cp のときには
 C++ プログラムとしてコンパイルします。ソースプログラムの拡張子を指定しなかった場合は、
 C プログラムとしてコンパイルします。

例 shc test.c C プログラムとしてコンパイルします。
 shc test.cpp C++ プログラムとしてコンパイルします。
 shc -lang=c++ test.c C++ プログラムとしてコンパイルします。
 shc test test.c を仮定し、C プログラムとしてコンパイルします。

備 考 lang=c を指定した場合、ecpp オプションは無効になります。

コピーライト出力制御

LOGO
NOLOGOなし(常に `nologo` が有効)書 式 `LOGO`
`NOLOGO`説 明 コピーライトの出力を抑止します。
`logo` オプションを指定した場合、コピーライト表示を出力します。
`nologo` オプションを指定した場合、コピーライト表示の出力を抑止します。
本オプション省略時解釈は、`logo` です。

文字列内文字コード

Euc
SJis
LATin1

なし

書 式 `Euc`
`SJis`
`LATin1`説 明 文字列、文字定数およびコメント内に日本語または ISO-Latin1 コードを記述できます。
ホストマシンと文字列内コードとの関係を表 2.11 に示します。

表 2.11 ホストマシンと文字列内コード

ホストマシン	オプション指定			
	<code>euc</code>	<code>sjis</code>	<code>latin1</code>	指定なし
PC	<code>euc</code>	<code>sjis</code>	<code>latin1</code>	<code>sjis</code>
SPARC	<code>euc</code>	<code>sjis</code>	<code>latin1</code>	<code>euc</code>
HP9000/700	<code>euc</code>	<code>sjis</code>	<code>latin1</code>	<code>sjis</code>

備 考 `latin1` オプションを指定した場合、`outcode` オプションは無効になります。

オブジェクト内漢字コード指定

OUtcode

なし

書 式 OUtcode = { EUc | SJis }

説 明 文字列、文字定数内に日本語を記述したときに、オブジェクトプログラムに出力する漢字コードを指定します。

outcode=euc は、漢字コードを euc コードで出力します。

outcode=sjis は、漢字コードを sjis コードで出力します。

ソースプログラム上の漢字コードは、euc または sjis オプションで指定できます。

サブコマンドファイル指定

SUbcommand

なし

書 式 SUbcommand = <ファイル名>

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

例 opt.sub : -show=object -debug
 コマンドライン指定 : shc -cpu=sh4 -subcommand=opt.sub test.c
 コンパイラ解釈 : shc -cpu=sh4 -show=object -debug test.c

3. アセンブラオプション

3.1 オプション指定規則

アセンブラを起動するコマンドラインの形式は以下のとおりです。

```
asmsh [ <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> [, ...]]
```

【注】* 複数のソースファイル名を指定すると、それらのファイルを指定順に連結したものがアセンブル処理の単位になります。この場合、.END は最後のファイルにだけ記述してください。

3.2 オプション解説

コマンドライン形式の英大文字は短縮形を、下線は省略時解釈を示します。

また、日立統合開発環境の対応するダイアログメニューをタブ名[項目]で示します。オプションの順序は日立統合開発環境のタブに対応しています。

3.2.1 Source オプション

表 3.1 Source カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 インクルード ファイル ディレクトリ	Include = <パス名>[, ...]	Source [Show entries for:] [Include file directories]	インクルードファイルの 取り込み先を指定します。
2 置換シンボル の定義	DEFine = <sub>[, ...] <sub> : <置換シンボル> = "<文字列>"	Source [Show entries for:] [Defines]	文字列の置き換えを定義 します。
3 整数型 プリプロセッサ 変数の定義	ASsignA = <sub>[, ...] <sub> : <変数名> = <整数定数>	Source [Show entries for:] [Preprocessor variables]	整数型のプリプロセッサ 変数を定義します。
4 文字型 プリプロセッサ 変数の定義	ASsignC = <sub>[, ...] <sub> : <変数名> = "<文字列>"	Source [Show entries for:] [Preprocessor variables]	文字型のプリプロセッサ 変数を定義します。

3. アセンブラオプション

インクルードファイルディレクトリ

Include

Source[Show entries for:][Include file directories]

書 式 Include = <パス名>[,...]

説 明 インクルードするファイルのディレクトリ名を指定します。
ディレクトリ名はホストマシンの標準的な指定方法に従います。
ディレクトリ名の指定数はコマンドラインで 1 行入力可能な限り有効です。
検索は、まずカレントディレクトリ、続いて include オプションで指定したディレクトリを
指定した順序に従って行います。

例 `asmsh aaa.src -include=C:¥common,C:¥local`
aaa.src 内で `.INCLUDE "file.h"` を指定した場合、file.h をカレントディレクトリ、
C:¥common、C:¥local の順に検索します。

備 考 アセンブラ制御文との関係

オプション	制御文	結 果
include	(指定に関わらず)	.INCLUDE で指定したディレクトリ include オプションで指定したディレクトリ
(指定なし)	.INCLUDE <ファイル名>	.INCLUDE で指定したディレクトリ

【注】* .INCLUDE で指定したディレクトリ文字列の前に include オプションで指定したディ
レクトリ文字列を付加したディレクトリ名を使用します。

置換シンボル定義

DEFine

Source[Show entries for:][Defines]

書 式 DEFine = <sub>[,...]
<sub> : <置換シンボル> = "<文字列>"

説 明 プリプロセッサで置換シンボルを対応する文字列に置き換えます。
define オプションと assignc オプションの機能の違いは、.DEFINE と .ASSIGNC の機能
の違いに対応します。

備 考 アセンブラ制御文との関係

オプション	制御文	結 果
define	.DEFINE (指定なし)	define オプションで指定した文字列 define オプションで指定した文字列
(指定なし)	.DEFINE	.DEFINE で指定した文字列

【注】* define オプションで置換シンボルに文字列を設定した場合、当該置換シンボルへの
.DEFINE による定義がすべて無効になります。

整数型プリプロセッサ変数定義

***A*SignA**

Source[Show entries for:][Preprocessor variables]

書 式 ASignA = <sub>[,...]
 <sub> : <プリプロセッサ変数名> = <整数定数>

説 明 プリプロセッサ変数に整数定数を設定します。
 プリプロセッサ変数名の書き方はシンボル名の書き方と同じです。
 整数定数は基数 (B'、Q'、D'、H') と数値を組み合わせで指定します。基数を省略し、数値のみを指定した場合は 10 進数として扱います。
 整数定数に指定できる値の範囲は -2,147,483,648 ~ 4,294,967,295 です。ただし、負の値を設定する場合は 10 進以外の基数で指定してください。

例 asmsh aaa.src -assigna=_\$=H'FF
 プリプロセッサ変数_\$に値 H'FF を設定します。ソースプログラム内のプリプロセッサ変数_\$のすべての参照箇所¥&_\$を H'FF に設定します。

備 考 ホスト OS が UNIX で、プリプロセッサ変数名に\$がある場合、もしくは基数表示のアポストロフィがある場合、直前にバックスラッシュ (円記号 "¥") を指定します。

アセンブラ制御文との関係

オプション	制御文	結 果
assigna	.ASSIGNA`	assigna オプションで指定した値
	(指定なし)	assigna オプションで指定した値
(指定なし)	.ASSIGNA	.ASSIGNA で指定した値

【注】* assigna オプションでプリプロセッサ変数に値を設定した場合、当該プリプロセッサへの.ASSIGNA による定義が無効になります。

A_SsignC

Source[Show entries for:][Preprocessor variables]

書 式 A_SsignC = <sub>[,...]
 <sub> : <プリプロセッサ変数名> = "<文字列>"

説 明 プリプロセッサ変数に文字列を設定します。
 プリプロセッサ変数名の書き方はシンボル名の書き方と同じです。
 文字列は文字をダブルコーテーション(")で囲んで指定します。
 文字列には 255 文字まで指定できます。

例 asmsh aaa.src -assignc=_\$="ON!OFF"
 プリプロセッサ変数_\$に文字列 ON!OFF を設定します。ソースプログラム内のプリプロセッサ変数_\$のすべての参照箇所%&_\$を文字列 ON!OFF に設定します。

備 考 ホスト OS が UNIX の場合は文字列の中に次の文字を指定する際、直前にバックスラッシュ(円記号 "\")を指定します。
 また、前後に文字列を指定する場合は前後の文字列をダブルコーテーション(")で囲みます。
 ・イクスクラメーション(!)
 ・ダブルコーテーション(")
 ・ドル(\$)
 ・逆コーテーション(`)

アセンブラ制御文との関係

オプション	制御文	結 果
assignc	.ASSIGNC	assignc オプションで指定した文字列
	(指定なし)	assignc オプションで指定した文字列
(指定なし)	.ASSIGNC	.ASSIGNC で指定した文字列

【注】* assignc オプションでプリプロセッサ変数に文字列を設定した場合、当該プリプロセッサ変数への.ASSIGNC による定義がすべて無効になります。

3.2.2 Object オプション

表 3.2 Object カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 デバッグ情報の出力制御	Debug NODebug	Object [Debug information:]	デバッグ情報の出力を制御します。
2 プリプロセッサの展開結果出力	EXPand [= <出力ファイル名>]	Object [Generate assembly source file after preprocess]	プリプロセッサの展開結果を出力します。
3 リテラルプールの出力ポイントの指定	LITERAL = <point> [, ...] <point> : { Pool Branch Jump Return }	Object [Generate literal pool after:]	リテラルプールを出力する場所を指定します。
4 オブジェクトモジュールの出力制御	Object[= <出力ファイル名>] NOObject	Object [Output file directory:]	オブジェクトモジュールの出力を制御します。

デバッグ情報

Debug
NODebug

Object[Debug information:]

書 式
Debug
NODebug

説 明
debug オプションを指定した場合、デバッグ情報を出力します。
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。
debug、nodebug オプションの指定は、オブジェクトモジュールを出力する場合に限り有効です。
本オプションの省略時解釈は、nodebug です。

備 考
デバッグ情報はデバッガでプログラムをデバッグするのに必要です。ソースプログラムの行に関する情報やシンボルに関する情報(シンボルデバッグ情報)などを含みます。

アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果 (オブジェクトモジュール出力時)
debug	(指定に関わらず)	デバッグ情報を出力する。
nodebug	(指定に関わらず)	デバッグ情報を出力しない。
(指定なし)	.OUTPUT DBG	デバッグ情報を出力する。
	.OUTPUT NODBG	デバッグ情報を出力しない。
	(指定なし)	デバッグ情報を出力しない。

プリプロセッサ展開結果出力

EXPand

	Object[Generate assembly source file after preprocess]
書 式	EXPand [= <出力ファイル名>]
説 明	<p>マクロ展開、条件つきアセンブル、ファイルのインクルードを行った後のアセンブリソースファイルを出力します。</p> <p>本オプションを指定するとオブジェクトの生成は行いません。</p> <p>出力ファイル名の指定を省略すると次のようになります。</p> <ul style="list-style-type: none">・ファイル拡張子の指定を省略した場合 ファイル拡張子は <code>exp</code> になります。・主ファイル名、ファイル拡張子とも指定を省略した場合 主ファイル名は入力ソースファイル(1 つめに指定したもの)と同じになります。 また、ファイル拡張子は <code>exp</code> になります。
備 考	入力ソースファイルと出力ファイルに、同じファイル名を指定しないでください。

リテラルプール出力ポイント指定

LITERAL

	Object[Generate literal pool after:]
書 式	<pre>LITERAL = <point>[,...] <point>: { Pool Branch Jump Return }</pre>
説 明	<p>リテラルプール自動生成機能によって生成されるリテラルプールの出力位置を指定します。</p> <ul style="list-style-type: none">・ <code>pool</code> <code>.POOL</code> の位置に出力します・ <code>branch</code> <code>BRA/BRAF</code> 命令の後に出力します・ <code>jump</code> <code>JMP</code> 命令の後に出力します・ <code>return</code> <code>RTS/RTE</code> 命令の後に出力します <p>本オプションの省略時解釈は、<code>literal=pool,branch,jump,return</code> です。</p>

オブジェクトモジュール出力制御

Object
NOObject

Object[Output file directory:]

書 式 Object [= <出力ファイル名>]
 NOObject

説 明 object オプションを指定した場合、オブジェクトモジュールを出力します。
 noobject オプションを指定した場合、オブジェクトモジュールを出力しません。
 出力ファイル名の指定を省略すると次のようになります。
 ・ファイル拡張子の指定を省略した場合
 ファイル拡張子は obj になります。
 ・主ファイル名、ファイル拡張子とも指定を省略した場合
 主ファイル名は入力ソースファイル(1 つめに指定したもの)と同じになります。
 また、ファイル拡張子は obj になります。
 本オプションの省略時解釈は、object です。

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果
object	(指定に関わらず)	オブジェクトファイルを出力する。
noobject	(指定に関わらず)	オブジェクトファイルを出力しない。
(指定なし)	.OUTPUT OBJ	オブジェクトファイルを出力する。
	.OUTPUT NOOBJ	オブジェクトファイルを出力しない。
	(指定なし)	オブジェクトファイルを出力する。

入力ソースファイルと出力オブジェクトファイルに、同じファイル名を指定しないでください。同じファイル名を指定した場合、入力ソースファイルが上書きされます。

3. アセンブラオプション

3.2.3 List オプション

表 3.3 List カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 アセンブルリスト の出力制御	LISt [= <出力ファイル名>] NOLISt	List [Generate list file]	アセンブルリストの出力を 制御します。
2 ソースプログラム リストの出力制御	SOUrce NOSOUrce	List [Source program:]	ソースプログラムリストの 出力を制御します。
3 ソースプログラム リストの部分出力 の制御	SHow [= <item>[, ...]] NOSHow [= <item>[, ...]] <item> : { CONditionals Definitions CALLs Expansions CODe }	List [Source program list Contents:] [Conditions:] [Definitions:] [Calls:] [Expansions:] [Code:]	ソースプログラムリストの 部分出力を制御します。
4 クロスリファレンス リストの出力制御	CRoss_reference NOCross_reference	List [Cross reference:]	クロスリファレンスリスト の出力を制御します。
5 セクション情報 リストの出力制御	SEction NOSEction	List [Section:]	セクション情報リストの 出力を制御します。

【注】* これらのオプションは、list オプションを指定した場合のみ有効となります。

アセンブルリスト出力制御

LIST
NOLIST

List[Generate list file]

書 式 LIST [= <出力ファイル名>]
NOLIST

説 明 list オプションを指定した場合、アセンブルリストを出力します。
nolist オプションを指定した場合、アセンブルリストを出力しません。
出力ファイル名の指定を省略すると次のようになります。

- ・ファイル拡張子の指定を省略した場合
ファイル拡張子は lis になります。
- ・主ファイル名、ファイル拡張子とも指定を省略した場合
主ファイル名は入力ソースファイル(1 つめに指定したもの)と同じになります。
また、ファイル拡張子は lis になります。

 本オプションの省略時解釈は、nolist です。

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果
list	(指定に関わらず)	アセンブルリストを出力する。
nolist	(指定に関わらず)	アセンブルリストを出力しない。
(指定なし)	.PRINT LIST	アセンブルリストを出力する。
	.PRINT NOLIST	アセンブルリストを出力しない。
	(指定なし)	アセンブルリストを出力しない。

入力ソースファイルと出力リストファイルに、同じファイル名を指定しないでください。
同じファイル名を指定した場合、入力ソースファイルが上書きされます。

3. アセンブラオプション

ソースプログラムリスト出力制御

*S*ource *N*O*S*ource

List[Source program:]

書 式 *S*ource
 *N*O*S*ource

説 明 *source* オプションを指定した場合、アセンブルリストにソースプログラムリストを付加します。
nosource オプションを指定した場合、アセンブルリストにソースプログラムリストを付加しません。
source、*nosource* オプションの指定は、アセンブルリストを出力する場合のみ有効です。本オプションの省略時解釈は、*source* です。

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果 (アセンブルリスト出力時)
<i>source</i>	(指定に関わらず)	ソースプログラムリストを出力する。
<i>nosource</i>	(指定に関わらず)	ソースプログラムリストを出力しない。
(指定なし)	.PRINT SRC	ソースプログラムリストを出力する。
	.PRINT NOSRC	ソースプログラムリストを出力しない。
	(指定なし)	ソースプログラムリストを出力する。

ソースプログラムリスト部分出力制御

SHow
NOSHow

List[Source program list Contents:]

[Conditions:], [Definitions:], [Calls:], [Expansions:], [Code:]

書 式 **SHow** [= <出力種別>[,...]]
 NOSHow [= <出力種別>[,...]]
 <出力種別>: { CONDitional | Definitions | CALLs | Expansions | CODE }

説 明 **show** オプションを指定した場合、ソースプログラムリストのプリプロセッサ機能のソースステートメント、オブジェクトコード表示行を出力します。**show**=<出力種別>を指定した場合は、指定した項目のみ出力します。
noshow オプションを指定した場合、ソースプログラムリストのプリプロセッサ機能のソースステートメント、オブジェクトコード表示行を出力しません。**noshow**=<出力種別>を指定した場合は、指定した項目のみ出力しません。
show、**noshow** オプションによる指定はアセンブルリストを出力する場合のみ有効です。出力種別の内容は次のとおりです。
 本オプションの省略時解釈は、**show** です。

出力種別	意 味	内 容
conditionals	条件つき不成立	.AIF, .AIFDEF の不成立部分
definitions	定義	マクロ定義部分 .AREPEAT, .AWHILE 定義部分 .INCLUDE .ASSIGNA, .ASSIGNC
calls	コール	マクロコール文 .AIF, .AIFDEF, .AENDI
expansions	展開	マクロ展開部分 .AREPEAT, .AWHILE 展開部分
code	オブジェクト コード表示行	制御命令のオブジェクトコード表示が、ソースステートメントの行数を超える部分

備 考 PC 版の場合、出力種別を 2 つ以上指定する時はカッコ()で囲んで指定してください。

アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果
show [=<出力種別>]	(指定に関わらず)	出 力
noshow [=<出力種別>]	(指定に関わらず)	出力抑止
(指定なし)	.LIST <出力種別>(出力)	出 力
	.LIST <出力種別>(出力抑止)	出力抑止
	(指定なし)	出 力

3. アセンブラオプション

クロスリファレンスリスト出力制御

Cross_reference *NOCross_reference*

List[Cross reference:]

書 式 Cross_reference
 NOCross_reference

説 明 cross_reference オプションを指定した場合、アセンブルリストにクロスリファレンスリストを付加します。
 nocross_reference オプションを指定した場合、アセンブルリストにクロスリファレンスリストを付加しません。
 cross_reference、nocross_reference オプションの指定は、アセンブルリストを出力する場合のみ有効です。
 本オプションの省略時解釈は、cross_reference です。

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果 (アセンブルリスト出力時)
cross_reference	(指定に関わらず)	クロスリファレンスリストを出力する。
nocross_reference	(指定に関わらず)	クロスリファレンスリストを出力しない。
(指定なし)	.PRINT CREF	クロスリファレンスリストを出力する。
	.PRINT NOCREF	クロスリファレンスリストを出力しない。
	(指定なし)	クロスリファレンスリストを出力する。

セクション情報リスト出力制御

SEction
NOSEction

List[Section:]

書 式 SEction
NOSEction

説 明 section オプションを指定した場合、アセンブルリストにセクション情報リストを付加します。

nosection オプションを指定した場合、アセンブルリストにセクション情報リストを付加しません。

section、nosection オプションの指定はアセンブルリストを出力する場合のみ有効です。本オプションの省略時解釈は、section です。

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果 (アセンブルリスト出力時)
section	(指定に関わらず)	セクション情報リストを出力する。
nosection	(指定に関わらず)	セクション情報リストを出力しない。
(指定なし)	.PRINT SCT	セクション情報リストを出力する。
	.PRINT NOSCT	セクション情報リストを出力しない。
	(指定なし)	セクション情報リストを出力する。

3. アセンブラオプション

3.2.4 Other オプション

表 3.4 Other カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 リテラルプール自動生成機能のサイズモードを指定	AUTO_literal	Other [Miscellaneous options:] [Automatically generate literal pool for immediate value]	リテラルプール自動生成機能をサイズ選択モードに設定します。

リテラルプール自動生成機能サイズモード設定

AUTO_literal

Other[Miscellaneous options:]

[Automatically generate literal pool for immediate value]

書 式 AUTO_literal

説 明 リテラルプール自動生成機能のサイズモードを指定します。
 本オプションを指定した場合、リテラルプール自動生成機能はサイズ選択モードとなり、オペレーションサイズ指定のないデータ転送命令(MOV #imm,Rn)はアセンブラが imm の値の範囲を判定し、必要ならばリテラルプールを自動生成します。
 本オプションを指定しない場合、リテラルプール自動生成機能はサイズ指定モードとなり、オペレーションサイズ指定がないデータ転送命令は 1 バイトのデータ転送命令としてアセンブルします。
 サイズ選択モードでは、オペレーションサイズ指定のないデータ転送命令は符号付きの範囲で判定するため、H'00000080 ~ H'000000FF(128 ~ 255)はワードサイズとして扱います。

imm の値の範囲	選択されるサイズまたはエラー	
	サイズ選択モード	サイズ指定モード
H'80000000 ~ H'FFFF7FFF (-2,147,483,648 ~ -32,769)	ロングワード	ウォーニング 835
H'FFFF8000 ~ H'FFFFFF7F (-32,768 ~ -129)	ワード	ウォーニング 835
H'FFFFFF80 ~ H'0000007F (-128 ~ 127)	バイト	バイト
H'00000080 ~ H'000000FF (128 ~ 255)	ワード	バイト
H'00000100 ~ H'00007FFF (256 ~ 32,767)	ワード	ウォーニング 835
H'00008000 ~ H'7FFFFFFF (32,768 ~ 2,147,483,647)	ロングワード	ウォーニング 835

【注】()内は 10 進表示

3.2.5 CPU オプション

表 3.5 CPU タブオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 CPU 種類の指定	CPU = <CPU 種別>	CPU [CPU:]	CPU 種別を指定します。
2 メモリのバイト並び順の指定	ENdian = { Big Little }	CPU [Endian:]	メモリのバイト並び順を指定します。
3 浮動小数点定数の丸め方式を指定	Round = { Nearest Zero }	CPU [Round to:]	浮動小数点定数の丸め方式を指定します。
4 非正規化数となる浮動小数点定数の取扱いを指定	DENormalize = { ON OFF }	CPU [Denormalized number allower as a result:]	浮動小数点定数が非正規化数となったときの取扱いを指定します。

CPU 種別**CPU**

CPU[CPU:]

書 式 CPU = <CPU 種別>

説 明 アセンブルするソースプログラムの対象とする CPU 種別を指定します。
CPU 種別の内容は次のとおりです。

CPU 種別	対象 CPU
SH1	SH-1 用
SH2	SH-2 用
SH2E	SH-2E 用
SH3	SH-3 用
SH4	SH-4 用
SHDSP	SH2-DSP 用
SH3DSP	SH3-DSP 用

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	環境変数	結 果
cpu=<CPU 種別> (指定なし)	(指定に関わらず) .CPU <CPU 種別> (指定なし)	(指定に関わらず) (指定に関わらず) SHCPU=<CPU 種別> (指定なし)	cpu オプションで指定した CPU 種別 .CPU で指定した CPU 種別 環境変数の CPU 種別 SH-1 用

3. アセンブラオプション

メモリのバイト並び順

ENdian

CPU[Endian:]

書 式 ENdian = { **Big** | Little }

説 明 ターゲットマイコンのバイトの並び方が、Big Endian か Little Endian かを指定します。本オプションの省略時解釈は、endian=big です。

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果 (アセンブルリスト出力時)
endian=big	(指定に関わらず)	Big Endian でアセンブルする。
endian=little	(指定に関わらず)	Little Endian でアセンブルする。
(指定なし)	.ENDIAN BIG	Big Endian でアセンブルする。
	.ENDIAN LITTLE	Little Endian でアセンブルする。
	(指定なし)	Big Endian でアセンブルする。

丸め方式

Round

CPU[Round to:]

書 式 Round = { Nearest | Zero }

説 明 浮動小数点データ制御命令に記述した定数をオブジェクトコードに変換する際の丸め方式を指定します。

round=nearest を指定した場合、round to NEAREST even で丸めます。

round=zero を指定した場合、round to ZERO で丸めます。

本オプションを省略した場合、CPU 種別により丸め方式は次のようになります。

CPU 種別	丸め方式
SH1	round to NEAREST even
SH2	round to NEAREST even
SH2E	round to ZERO
SH3	round to NEAREST even
SH4	round to ZERO
SHDSP	round to NEAREST even
SH3DSP	round to NEAREST even

備 考 CPU 種別が SH2E で、丸め方式に round to NEAREST even を選択した場合、ソースプログラムで最初に出現した浮動小数点データ制御命令に対してウォーニング 818 とし、round to NEAREST even でオブジェクトコードを出力します。

非正規化数の扱い

DENormalize

CPU[Denormalize:]

書 式 DENormalize = { ON | OFF }

説 明 浮動小数点データ制御命令に非正規化数を記述したとき有効な値とするか、無効な値とするかを指定します。非正規化数を有効な値(ON)とした場合と、無効な値(OFF)とした場合ではオブジェクトコードが異なります。

denormalize=onを指定した場合、非正規化数を有効な値とします。

denormalize=offを指定した場合、非正規化数を無効な値とします。

非正規化数を有効とした場合は、ウォーニング 842 とし、オブジェクトコードを出力します。

非正規化数を無効とした場合は、ウォーニング 841 とし、0 のオブジェクトコードを出力します。

本オプションを省略した場合、CPU 種別により非正規化数の扱いは次のようになります。

CPU 種別	有効 / 無効
SH1	有効な値(ON)とする
SH2	有効な値(ON)とする
SH2E	無効な値(OFF)とする
SH3	有効な値(ON)とする
SH4	無効な値(OFF)とする
SHDSP	有効な値(ON)とする
SH3DSP	有効な値(ON)とする

備 考 CPU 種別が SH2E で、非正規化数の扱いを有効な値とした場合、ソースプログラムで最初に出現した浮動小数点データ制御命令に対してウォーニング 818 とし、非正規化数の扱いを有効な値としてオブジェクトコードを出力します。

3. アセンブラオプション

3.2.6 その他のオプション

表 3.6 その他のオプション

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 異常終了とするエラーレベルの変更	ABort = { Warning Error }	-	アセンブラが異常終了するエラーのレベルを変更します。
2 欧州コード文字	LATIN1	-	ソースファイル内に欧州コード文字を使えるようにします。
3 漢字コードをシフト JIS に指定	SJIS	-	ソースファイル内の漢字コードをシフト JIS コードとして扱います。
4 漢字コードを EUC に指定	EUC	-	ソースファイル内の漢字コードを EUC コードとして扱います。
5 出力漢字コードの指定	OUtcode = { SJIS EUC }	-	オブジェクトファイルに出力する漢字コードを指定します。
6 アセンブルリストの行数指定	LINEs = <行数>	-	アセンブルリストの行数を設定します。
7 アセンブルリストの桁数指定	COlumnS = <桁数>	-	アセンブルリストの桁数を設定します。
8 コピーライト出力抑止	LOGO NOLOGO	- (常に nologo が有効)	コピーライトを出力します コピーライトの出力を抑止します
9 サブコマンドファイルの指定	SUBcommand = <ファイル名>	-	コマンドラインをファイルから入力します。

エラーレベル変更

ABort

なし

書 式 ABort = { Warning | Error }

説 明 異常終了するエラーレベルを変更します。
 abort=warning を指定した場合、ウォーニングが発生すると処理を中断します。
 abort=error を指定した場合、エラーが発生すると処理を中断します。
 OS へのリターン値が 1 以上の場合、オブジェクトモジュールの出力を抑止します。
 abort オプションによる指定はオブジェクトモジュールを出力する場合に限り有効です。
 OS へのリターン値は次のとおりです。
 本オプションの省略時解釈は、abort=error です。

発生回数			オプション指定時の OS へのリターン値			
ウォーニング	エラー	致命的エラー	abort=warning		abort=error	
			PC	UNIX	PC	UNIX
0	0	0	0	0	0	0
1 以上	0	0	2	1	0	0
-	1 以上	0	2	1	2	1
-	-	1 以上	4	1	4	1

欧州コード文字

LATINI

なし

書 式 LATIN1

説 明 文字列およびコメント内に欧州コード文字を記述することができます。
本オプションは、`sjis`、`euc`、または `outcode` オプションと一緒に指定しないでください。

シフト JIS

SJIS

なし

書 式 SJIS

説 明 文字列およびコメント内の日本語をシフト JIS コードとして解釈します。
本オプション、`euc` オプションとも省略した場合は、文字列、コメント内の日本語はホストマシンに依存する日本語コードとして解釈します。
本オプションは、`latin1`、または `euc` オプションと一緒に指定しないでください。

3. アセンブラオプション

EUC

EUC

なし

書 式 `EUC`

説 明 文字列およびコメント内の日本語を `EUC` コードとして解釈します。
本オプション、`sjis` オプションとも省略した場合は、文字列、コメント内の日本語はホストマシンに依存する日本語コードとして解釈します。
本オプションは、`latin1`、または `sjis` オプションと一緒に指定しないでください。

出力漢字コード

OUtcode

なし

書 式 `OUtcode = { SJIS | EUC }`

説 明 `outcode=sjis` を指定した場合、ソースファイル内の日本語記述をシフト JIS コードに変換し、オブジェクトファイルに出力します。
`outcode=euc` を指定した場合、ソースファイル内の日本語記述を `EUC` コードに変換し、オブジェクトファイルに出力します。
本オプションとソースファイル内の漢字コードの指定 (`sjis`、`euc` オプション) によるオブジェクトファイルへ出力する漢字コードは次のとおりです。

outcode オプション	ソースファイル内の漢字コード		
	<code>sjis</code>	<code>euc</code>	指定なし
<code>sjis</code>	シフト JIS コード	シフト JIS コード	シフト JIS コード
<code>euc</code>	<code>EUC</code> コード	<code>EUC</code> コード	<code>EUC</code> コード
指定なし	シフト JIS コード	<code>EUC</code> コード	デフォルト漢字コード

本オプションを省略した場合の漢字コードは次のとおりです。

ホスト	漢字コード
SPARC ステーション	<code>EUC</code> コード
HP9000/700 シリーズ	シフト JIS コード
PC	シフト JIS コード

アセンブルリスト行数設定

LINes

なし

書 式 LINes = <行数>

説 明 アセンブルリストの1ページあたりの行数を設定します。
 行数として有効な値は20～255です。
 本オプションはアセンブルリストを出力する場合のみ有効です。

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果
lines=<行数>	(指定に関わらず)	1ページあたり、lines オプションで指定した 行数になる。
(指定なし)	.FORM LIN=<行数>	1ページあたり、.FORM で指定した行数になる。
	(指定なし)	1ページあたり、60行になる。

アセンブルリスト桁数設定

COlumns

なし

書 式 COlumns = <桁数>

説 明 アセンブルリストの1行あたりの桁数を設定します。
 桁数として有効な値は79～255です。
 本オプションはアセンブルリストを出力する場合のみ有効です。

備 考 アセンブラ制御命令との関係(アセンブラはオプションによる指定を優先します)

オプション	制御命令	結 果
columns=<桁数>	(指定に関わらず)	1行あたり、columns オプションで指定した 桁数になる
(指定なし)	.FORM COL=<桁数>	1行あたり、.FORM で指定した桁数になる。
	(指定なし)	1行あたり、132桁になる。

コピーライト出力制御

LOGO
NOLOGO

なし(常に `nologo` が有効)

書 式 `LOGO`
 `NOLOGO`

説 明 コピーライトの出力を制御します。
 `logo` オプションを指定した場合、コピーライト表示が出力されます。
 `nologo` オプションを指定した場合、コピーライトの表示の出力が抑止されます。
 本オプション省略時解釈は、`logo` です。

サブコマンドファイル指定

SUBcommand

なし

書 式 `SUBcommand = <ファイル名>`

説 明 コマンドラインをファイルから入力します。
 通常のコマンドライン指定と同じ順番で、入力ファイル名とオプションを指定してください。
 1行に1つの入力ファイル名またはオプションを指定してください。
 `subcommand` オプションは、サブコマンドファイル内に指定しないでください。

例 `asmsh aaa.src -subcommand=aaa.sub`
 サブコマンドファイルの内容をコマンドラインに展開し、アセンブルします。
 `aaa.sub` の内容：
 `bbb.src`
 `-list`
 `-noobj`
 展開結果：
 `asmsh aaa.src,bbb.src -list -noobj`

備 考 サブコマンドファイル全体のサイズの限界値は 65,535 バイトです。

4. 最適化リンケージエディタ操作方法

4.1 オプション指定規則

4.1.1 コマンドラインの形式

コマンドラインの形式は以下のとおりです。

```
optlnk [ { <ファイル名> | <オプション列>}...]  
      <オプション列>: --<オプション>[= <サブオプション>[,...]]
```

4.1.2 サブコマンドファイルの形式

サブコマンドファイルの形式は以下のとおりです。

```
<オプション> {= | }[<サブオプション>[,...]] [ &] [ ;<コメント>]  
&: 継続行指定
```

サブコマンドファイル形式の詳細は、「4.2.7 subcommand file オプション」を参照してください。

4.2 オプション解説

オプション、サブオプションの英大文字は短縮形指定時の文字を、下線は省略時解釈を示します。
 また、日立統合開発環境の対応するダイアログメニューを、タブ名[項目]で示します。オプションの順序は、日立統合開発環境のタブに対応しています。

4.2.1 Input オプション

表 4.1 Input カテゴリオプション一覧

項目	オプション	ダイアログメニュー	指定内容
1 入力 ファイル	Input = <sub> [{, }...] <sub>: <ファイル名> [[<モジュール名>[...]]]	Input [Show entries for:] [Relocatable files and object files]	入力ファイルを指定 (コマンドラインでは input オプションなしで 指定します)
2 ライブラリ ファイル	LIBrary = <ファイル名>[...]	Input [Show entries for:] [Library files]	入力ライブラリファイルを 指定
3 バイナリ ファイル	Binary = <sub>[...] <sub> : <ファイル名> (<セクション名> [,<シンボル名>])	Input [Show entries for:] [Binary files]	入力バイナリファイルを指定
4 シンボル 定義	DEFine = <sub>[...] <sub>: <シンボル名> = { <シンボル名> <数値> }	Input [Show entries for:] [Defines:]	未定義シンボルの強制定義 シンボル名と同値として定義 数値で定義
5 実行開始 アドレス	ENTry = { <シンボル名> <アドレス> }	Input [Use entry point:]	エントリシンボルを指定 エントリアドレスを指定
6 プレリンカ 起動抑止	NOPRElink	Input [Prelinker control:]	プレリンカの起動を抑止

入力ファイル

Input

	Input[show entries for:][Relocatable files and object files]
書 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>[,...])]
説 明	<p>入力ファイルを指定します。複数ある場合にはカンマ(,)またはスペースで区切って指定します。</p> <p>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。</p> <p>入力ファイル名に拡張子の指定がない場合は、モジュール名指定なしの時は「obj」、モジュール名指定ありの時は「lib」を仮定します。</p>
例	<pre>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 を全て入力します</pre>
備 考	form=object および extract オプションを指定した場合、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。

ライブラリファイル

LIBrary

	Input[show entries for:][Library files]
書 式	LIBrary = <ファイル名>[,...]
説 明	<p>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。</p> <p>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。</p> <p>form=library 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。</p> <p>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。</p> <p>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。</p>
例	<pre>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 を全て入力します。</pre>

Binary

Input[show entries for:][Binary files]

- 書式** Binary = <sub>[,...]
<sub> : <ファイル名>(<セクション名>[, <シンボル名>])
- 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。
- 例** `input=a.obj`
`start=P,D*/200`
`binary=b.bin(D1bin),c.bin(D2bin,_datab)`
- `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,rellocate,library` または `strip` オプションを指定した場合、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。

シンボル定義**DEFine**

Input[show entries for:][Defines:]

- 書式** DEFine = <sub>[,...]
<sub> : <シンボル名> = { <シンボル名> | <数値> }
- 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,rellocate,library` を指定した場合、本オプションは無効です。

実行開始アドレス

ENTry

Input[Use entry point:]

書式	ENTry = { <シンボル名> <アドレス> }	
説明	<p>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。</p> <p>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。</p>	
例	<pre>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。</pre>	
備考	<p>form=object,rellocate,libraryまたはstripオプションを指定した場合、本オプションは無効です。</p> <p>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。</p>	

プレリンカ起動抑止

NOPRElink

Input[Prelinker control:]

書式	NOPRElink	
説明	<p>プレリンカの起動を抑止します。</p> <p>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、本オプションを指定してください。リンク速度を速くすることができます。</p>	
備考	<p>extractまたはstripオプションを指定した場合、本オプションは無効です。</p>	

4.2.2 Output オプション

表 4.2 Output カテゴリオプション一覧

項目	オプション	ダイアログメニュー	指定内容
1 出力形式	FOrm = { Absolute Relocate Object Library [= { S U }] Hexadecimal Stype Binary }	Output [Type of output file:]	アブソリュート形式 リロケータブル形式 オブジェクト形式 ライブラリ形式 HEX 形式 S タイプ形式 バイナリ形式
2 デバッグ 情報	DEBug SDeBug NODeBug	Output [Debug information:]	出力あり(出力ファイル内) デバッグ情報ファイル出力 出力なし
3 レコード サイズ統一	REcord = { H16 H20 H32 S1 S2 S3 }	Output [Data record header:]	HEX レコード 拡張 HEX レコード 32bitHEX レコード S1 レコード S2 レコード S3 レコード
4 ROM 化 支援	ROm = <sub>[...] <sub> : <ROM セクション名> =<RAM セクション名>	Output [Show entries for:] [ROM to RAM mapped sections:]	RAM に領域を確保し、シンボル を RAM 上のアドレスでリロケー ション解決
5 出力 ファイル	OUtput = <sub>[...] <sub> : <ファイル名> [=<出力範囲>] <出力範囲>: { <先頭アドレス> - <終了アドレス> <セクション名>[: ...] }	Output [Show entries for:] [Divided output files:]	出力ファイルを指定 (範囲指定、分割出力可能)
6 インフォ メーション メッセージ	Message NOMessage [= <sub>[...]] <sub> : <エラー番号> [- <エラー番号>]	Output [Show entries for:] [Messages:]	出力あり 出力なし (エラー番号、範囲指定可能)
7 外部シンボ ル割り付け 情報ファイ ル	MAP [= <ファイル名>]	Output [Generate map file]	外部シンボル割り付け情報ファ イル出力を指定

出力形式

FOrm

Output[Type of output file:]

```
書 式   FOrm = { Absolute      |
           Relocate          |
           Object            |
           Library[={S|U}]  |
           Hexadecimal       |
           Stype             |
           Binary            |
           }                  }
```

説 明 出力形式を指定します。
 本オプションの省略時解釈は、form=absolute です。
 サブオプションの一覧を表 4.3 に示します。

表 4.3 form オプションのサブオプション一覧

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	HEX ファイルを出力します。HEX フォーマットは「18.1.2 HEX フォーマット」を参照してください。
6 stype	S タイプファイルを出力します。S タイプフォーマットは「18.1.1 S タイプフォーマット」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 4.4 に示します。

4. 最適化リンケージエディタ操作方法

表 4.4 出力形式と入力ファイル、他オプションとの関係

出力形式	指定オプション	入力可能なファイル形式	指定可能なオプション ^{*1}
1 Absolute	strip あり	アブソリュートファイル	input, output, show=symbol,reference
	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, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, variable_forbid, function_forbid, absolute_forbid, profile, cachesize, rename, delete, define, fsymbol, stack, noprelink, show=symbol,reference
2 Relocate	extract あり	ライブラリファイル	library, output, show=symbol,reference
	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, rename, delete, noprelink, show=symbol,reference
3 Object	extract あり	ライブラリファイル	library, output, show=symbol,reference
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, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, variable_forbid, function_forbid, absolute_forbid, profile, cachesize, rename, delete, define, fsymbol, stack, noprelink, record, s9 ^{*2} , show=symbol,reference
		アブソリュートファイル	input, output, record, s9 ^{*2} , show=symbol,reference
5 Library	strip あり	ライブラリファイル	library, output, show=symbol,section
	extract あり	ライブラリファイル	library, output, show=symbol,section
	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, rename, delete, replace, noprelink, show=symbol,section

【注】 *1: message/nomessage, change_message, logo/nologo, form, list, subcommand は常に指定できます。
*2: s9 は出力形式が form=stype のときだけ指定できます。

デバッグ情報

DEBug
SDEbug
NODEBug

Output[Debug information:]

書式	DEBug SDEbug NODEBug
説明	debug 情報の出力有無を指定します。 debug オプションを指定した場合、出力ファイル中にデバッグ情報を出力します。 sdebug オプションを指定した場合、<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。 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。 form=relocate 指定時に sdebug オプションを指定した場合は、debug オプションと解釈します。 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場合は、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。 本オプション省略時解釈は、debug です。
備考	form=object,library,hexadecimal,stype,binary または strip,extract オプションを指定した場合、本オプションは無効です。

レコードサイズ統一

REcord

Output[Data record header:]

書式	REcord = { H16 H20 H32 S1 S2 S3 }
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。 本オプション省略時は、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。
備考	form=hexadecimal または stype オプションを指定していない場合、本オプションは無効です。

ROm

	Output[Show entries for:][ROM to RAM mapped sections]
書 式	ROm = <sub>[,...] <sub> : <ROM セクション名>=<RAM セクション名>
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にリロケーションします。 ROM セクションには初期値のあるリロケータブルセクションを指定します。 RAM セクションには存在しないセクションまたはサイズ 0 のリロケータブルセクションを指定します。
例	rom=D=R start=D/100,R/8000 D セクションと同サイズの R セクションを確保し、D セクション内定義シンボルを R セクション上のアドレスでリロケーションします。
備 考	form=object,relocate,library または strip オプションを指定した場合、本オプションは無効です。

出力ファイル**OOutput**

	Output[Show entries for:][Divided output files]
書 式	OOutput = <sub>[,...] <sub> : <ファイル名>[=<出力範囲>] <出力範囲>: { <先頭アドレス> - <終了アドレス> <セクション名>[:...] }
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。 本オプションの省略時解釈は、<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=hexadecima : 「hex」、form=sttype : 「mot」 form=binaray : 「bin」
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。

インフォメーションメッセージ

Message
NOMessage

Output[Show entries for:][Messages]

- 書式** Message
 NOMessage [= <sub>[,...]]
 <sub> : <エラー番号>[- <エラー番号>]
- 説明** インフォメーションレベルメッセージの出力有無を指定します。
 message オプション指定時は、インフォメーションレベルメッセージを出力します。
 nomessage オプション指定時は、インフォメーションレベルメッセージの出力を抑制します。
 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制できます。ハイフン(-)を使用して抑制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 オプション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制します。
 本オプションの省略時解釈は、nomessage です。
- 例** nomessage=4,200-203,1300
 L0004 および L0200 ~ L0203 および L1300 のメッセージ出力を抑制します。

外部シンボル割り付け情報ファイル出力

MAP

Output[Generate map file]

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

4. 最適化リンケージエディタ操作方法

4.2.3 List オプション

表 4.5 List カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 リスト ファイル	LISt [= <ファイル名>]	List [Generate list file:]	リストファイル出力を指定
2 リスト 内容	SHow [= <sub>[,...]] <sub> : { SYmbol Reference SEction }	List [Contents:]	シンボル情報 参照回数 セクション情報

リストファイル

LISt

List[Generate list file]

書 式 LISt [= <ファイル名>]

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`」、それ以外るとき「`map`」の
リストファイルが作成されます。

SHow

List[Contents]

書 式 SHow[= <sub>[,...]]
 <sub> : { SYmbol | Reference | SEction }

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 4.6 に示します。
 各リストの具体例については「8.4 リンケージリストの参照方法」、
 「8.5 ライブラリリストの参照方法」を参照してください。

表 4.6 show オプションのサブオプション一覧

	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。
	または	reference	指定できません。
	extract 指定時	section	モジュール内セクション一覧を出力します。
2	form=library 以外	symbol	シンボルアドレス、サイズ、種別、最適化内容を出力します。
	または	reference	シンボルの参照回数を出力します。
	extract 指定なし時	section	指定できません。

備 考 form=object,relocate を指定した場合、show=reference は無効です。

4.2.4 Optimize オプション

表 4.7 Optimize カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 最適化	OOptimize [= <sub>[,...]] <sub>: { STring_unify SYmbol_delete Variable_access Register SAMe_code Function_call Branch SPeed SAFe } NOOptimize	Optimize [Show entries for:] [Optimize items:] [Optimize:]	最適化あり 定数/文字列の統合 未参照シンボルの削除 短絶対アドレッシングモード活用 レジスタ退避・回復の最適化 共通コードの統合 間接アドレッシングモード活用 分岐命令の最適化 スピード重視の最適化 安全な最適化 最適化なし
2 共通コード サイズ	SAMESize = <サイズ> (省略時: <u>sames=1e</u>)	Optimize [Show entries for:] [Optimize items:] [Eliminated size:]	共通コード統合の対象となる最小 サイズの指定
3 プロファイ ル情報	PROfile = <ファイル名>	Optimize [Show entries for:] [Optimize items:] [Include profile:]	プロファイル情報ファイルの指定 (動的最適化を行います)
4 キャッシュ サイズ	CAchesize = Size = <サイズ>, Align = <ラインサイズ> (省略時: <u>ca=s=8,a=20</u>)	Optimize [Show entries for:] [Optimize items:] [Cache size:]	キャッシュサイズの指定 キャッシュラインサイズの指定
5 最適化 部分抑止	SYmbol_forbid = <シンボル名>[,...] SAMECode_forbid = <関数名>[,...] Variable_forbid = <シンボル名>[,...] FUunction_forbid = <関数名>[,...] Absolute_forbid = <アドレス> [+ <サイズ>] [...] 	Optimize [Show entries for:] [Forbid item:]	未参照シンボル削除抑止シンボル 共通コード統合抑止シンボル 短絶対アドレッシングモード活用 抑止シンボル 間接アドレッシングモード活用抑 止シンボル 最適化抑止アドレス範囲

OPTimize NOOPTimize

	Optimize[Show entries for:][Optimize items:][Optimize:]
書式	Optimize[= <sub>[,...]] NOOPTimize <sub> : { SString_unify SYmbol_delete Variable_access Register SAME_code Function_call Branch Speed SAFE }
説明	モジュール間最適化実行有無を指定します。 optimize オプションを指定した場合、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。 nooptimize オプションを指定した場合、モジュール間最適化を行いません。 本オプションの省略時解釈は、optimize です。 サブオプションの一覧を表 4.8 に示します。

表 4.8 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 ¹⁾			
		SHC	SHA	H8C	H8A
パラメタなし	全ての最適化を実行します。		x		
string_unify	const 属性を持つ定数に対し、同一値定数を統合します。 const 属性を持つ定数には次のものが含まれます。 ・ C/C++プログラム中で const 宣言した変数 ・ 文字列データの初期値 ・ リテラル定数		x		x
symbol_delete	1 度も参照のない変数 / 関数を削除します。必ず entry オプションを指定してください。		x		x
variable_access	8/16 ビット絶対アドレッシングモードでアクセス可能な領域にアクセス回数の多い変数を割り当てます。必ず cpu オプションを指定してください。	x	x		
register	関数の呼び出し関係を解析し、レジスタの再割付および冗長なレジスタ退避・回復コードを削除します。必ず entry オプションを指定してください。		x		x
same_code	複数の同一命令列をサブルーチン化します。		x		x
function_call	0 ~ 0xFF の範囲に空きがあれば、アクセス回数の多い関数のアドレスを割り当てます。必ず cpu オプションを指定してください。	x	x		
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。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。		x		
speed	オブジェクトスピード低下を招く可能性のある最適化以外を実行します。 optimize=string_unify,symbol_delete,variable_access,register,branch と同じです。		x		
safe	変数や関数の属性によって制限される可能性のある最適化以外を実行します。 optimize=string_unify,register,branch と同じです。		x		

【注】*1: SHC: SH 用 C/C++プログラム、SHA: SH 用アセンブリプログラム、
H8C: H8 用 C/C++プログラム、H8A: H8 用アセンブリプログラム

備考 form=object,rellocate,library または strip オプションを指定した場合、本オプションは無効です。

SAMESize

	Optimize[Show entries for:][Optimize items:][Eliminated size:]
書式	SAMESize = <サイズ>
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。0x8 ~ 0x7FFF までの 16 進数で指定してください。 本オプションの省略時解釈は、samesize=1E です。
備考	optimize=same_code を指定しない場合、本オプションは無効です。

PROfile

	Optimize[Show entries for:][Optimize items:][Include profile:]
書式	PROfile = <ファイル名>
説明	プロファイル情報ファイルを指定します。 プロファイル情報ファイルとして指定できるのは、HDI (Hitachi Debugging Interface) Ver5.0 以降が出力するプロファイル情報ファイルだけです。 プロファイル情報ファイルを指定すると、モジュール間最適化で動的情報に基づいた最適化を実行することができます。 プロファイル情報入力により影響がある最適化を表 4.9 に示します。

表 4.9 プロファイル情報と最適化の関係

サブオプション	意味	最適化対象プログラム ^{*1}			
		SHC	SHA	H8C	H8A
variable_access	動的アクセス回数の多い変数を優先的に割り当てます。	×	×		
function_call	動的アクセス回数の多い関数の最適化優先順位を下げます。	×	×		
branch	動的に呼び出し回数の多い関数を呼び出し元の関数の近くに配置します。 SH 用プログラムの場合は、cachesize オプションで指定するキャッシュサイズを意識した配置最適化を行います。				^{*2}

【注】 *1 SHC: SH 用 C/C++ プログラム、SHA: SH 用アセンブリプログラム、
H8C: H8 用 C/C++ プログラム、H8A: H8 用アセンブリプログラム

*2 関数単位の移動は行いませんが、入力ファイル単位の移動は実行します。

備考 optimize オプションを指定しない場合、本オプションは無効です。

キャッシュサイズ

CAcheseize

Optimize[Show entries for:][Optimize items:][Cache size:]

書 式 CAcheseize = Size = <サイズ>, Align = <ラインサイズ>

説 明 キャッシュサイズおよびキャッシュラインサイズを指定します。
 profile オプションを指定した場合に、分岐命令最適化(optimize=branch)で使
 用します。サイズは kbyte 単位、ラインサイズは byte 単位の 16 進数で指
 定してください。本オプションの省略時解釈は、cachesize=size=8,align=20
 です。

備 考 profile オプションを指定しない場合、本オプションは無効です。

最適化部分抑止

SYmbol_forbid***SAMECode_forbid******Variable_forbid******FUnction_forbid******Absolute_forbid***

Optimize[Show entries for:][Forbid item:]

書 式 SYmbol_forbid = <シンボル名>[,...]
 SAMECode_forbid = <関数名>[,...]
 Variable_forbid = <シンボル名>[,...]
 FUnction_forbid = <関数名>[,...]
 Absolute_forbid = <アドレス>[+ <サイズ>][,...]

説 明 特定のシンボル、アドレス範囲の最適化を抑止します。アドレス、サイズは 16 進数で指
 定してください。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。オプションの一覧を表
 4.10 に示します。

表 4.10 最適化部分抑止オプション一覧

オプション	パラメタ	意味
symbol_forbid	関数名 変数名	未参照シンボル削除最適化を抑止します。
samecode_forbid	関数名	共通コード統合最適化を抑止します。
variable_forbid	変数名	短絶対アドレッシングモード活用最適化を抑止します。
function_forbid	関数名	間接アドレッシングモード活用最適化を抑止します。
absolute_forbid	アドレス[+ サイズ]	アドレス+サイズの範囲の最適化を抑止します。

例 symbol_forbid="f(int)" ; C++関数 f(int) は参照回数 0 でも削除しません。

備 考 optimize オプションを指定しない場合、本オプションは無効です。

4.2.5 Section オプション

表 4.11 Section カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 セクション 開始アドレス	STAR t= <sub>[,...] <sub> : <セクション名> [{: ,} <セクション名>[,...]] [/<アドレス>]	Section [Show entries for:] [Section:]	セクションの開始アドレス指定
2 シンボル アドレス ファイル	FSymbol = <セクション名>[,...]	Section [Show entries for:] [Symbol file:]	外部定義シンボルアドレスの 定義ファイル出力

セクション開始アドレス

STARt

Section[Show entries for:][Section:]

書 式 STARt = <sub>[,...]
 <sub> : <セクション名>[{: |,} <セクション名>[,...]][/<アドレス>]

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。
 コロンの(:)で区切ることにより、同一アドレスへの複数セクションの割り付けを指定することもできます。
 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入力順に展開します。
 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。
 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 番地に割り付けます。
 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 start=P,C,D*/100,R1:R2/8000 ; D で始まるセクションに D1,D2 があると仮定
 ROM=D1=R1,D2=R2
 P,C,D1,D2 の順に 0x100 番地から割り付けます。R1,R2 はどちらも 0x8000 番地に割り付けます。

 input=a.obj b.obj ; a.obj は d.lib 内シンボル,b.obj は c.lib 内シンボルを参照
 library=c.lib,d.lib
 start=P/100
 P セクション内割り付け順序は、a(P),b(P),c(P),d(P)になります。

備 考 form=object,relocate,library または strip オプションを指定した場合、本オプションは無効です。

シンボルアドレスファイル

FSymbol

Section[Show entries for:][Symbol file:]

- 書 式** `FSymbol = <セクション名>[,...]`
- 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**
- ```
fsymbol=sct2,sct3
output=test.abs
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。
```
- [test.fsy の出力例]
- ```
;HITACHI OPTIMIZING LINKAGE EDITOR GENERATED FILE 1999.11.26
;fsymbol = sct2, sct3

;SECTION NAME = sct2
.export _f
_f: .equ h'00000000
.export _g
_g: .equ h'00000016
;SECTION NAME = sct3
.export _main
_main: .equ h'00000020
.end
```
- 備 考** `form=object,relocate,library` または `strip` オプションを指定した場合、本オプションは無効です。

4.2.6 Verify オプション

表 4.12 Verify カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 アドレス 整合性の チェック	CPu = { <cpu 情報ファイル名> {ROm RAm} = <アドレス範囲>[,...] } <アドレス範囲>: <先頭アドレス> - <終了アドレス>	Verify 【MAP/CPU information check:】	cpu 情報ファイルを指定 セクションアドレスの割り付け 可能範囲を指定

アドレス整合性チェック

CPu

	Verify【MAP/CPU information check:】
書 式	CPu = { <cpu 情報ファイル名> { ROm RAm } = <アドレス範囲>[,...] } <アドレス範囲> : <先頭アドレス> - <終了アドレス>
説 明	セクションの割り付けアドレスの整合性をチェックします。 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。 旧製品添付の cia(cpu information analyzer) で作成した cpu 情報ファイルを指定することもできます。
例	cpu=ROM=0-FFFF, RAM=10000-1FFFF セクションアドレスが、0x0-0xFFFF または 0x10000-0x1FFFF 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。
備 考	form=object, relocate, library または strip オプションを指定した場合、本オプションは無効です。

4.2.7 Other オプション

表 4.13 Other カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 終端コード	S9	Other [Miscellaneous options:] [Always output S9 record at the end]	S9 レコードを常に出力
2 スタック 情報 ファイル	STACK	Other [Miscellaneous options:] [Stack information output]	スタック使用量情報ファイル出力
3 デバッグ情 報圧縮	COmpress NOCOmpress	Other [Miscellaneous options:] [compress debug information]	デバッグ情報を圧縮する デバッグ情報を圧縮しない
4 シンボル名 変更	REName = <sub>[,...] <sub> : { [<ファイル名> (<名前>=<名前>[,...])] [<モジュール名> (<名前>=<名前>[,...])] }	-	シンボル名、セクション名の変更
5 シンボル名 削除	DElete = <sub>[,...] <sub> : { <モジュール名> [<ファイル名> (<名前>[,...])] }	-	シンボル名、モジュール名の削除
6 モジュール 置き換え	REPlace = <sub>[,...] <sub> : <ファイル> [(<モジュール>[,...])]	-	ライブラリファイル内同名 モジュールの置き換え
7 モジュール 抽出	EXtract = <モジュール>[,...]	-	ライブラリファイル内指定 モジュールの抽出
8 デバッグ 情報削除	STRip	-	アブソリュートファイル、 ライブラリファイルの デバッグ情報削除
9 メッセージ レベル	CHange_message = <sub>[,...] <sub>: { Information Warning Error } [=<エラー番号> [-<エラー番号>] [...]]	-	メッセージレベルの変更
10 コピーライ ト出力制御	LOgo NOLOgo	-	出力あり 出力なし
11 継続指定	END	-	既入力オプション列を実行し、処 理終了後は以降のオプション列を 入力し、処理を継続
12 終了指定	EXIt	-	オプション入力の終了を指定

S9

	<code>Other[Miscellaneous options:][Always output S9 record at the end]</code>
書 式	S9
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	<code>form=stype</code> を指定しない場合、本オプションは無効です。

STACK

	<code>Other[Miscellaneous options:][Stack information output]</code>
書 式	STACK
説 明	スタック使用量情報ファイルを出力します。 ファイル名は、<出力ファイル名>.sni になります。
備 考	<code>form=obj,rellocate,library</code> および <code>strip</code> オプションを指定した場合、本オプションは無効です。

デバッグ情報圧縮

***C*ompress
*NO*Compress**

Other[Miscellaneous options:][compress debug information]

書式	<code>C</code> ompress <code>NO</code> Compress
説明	デバッグ情報の圧縮有無を指定します。 <code>compress</code> オプションを指定した場合、デバッグ情報を圧縮します。 <code>nocompress</code> オプションを指定した場合、デバッグ情報を圧縮しません。 デバッグ情報を圧縮すると、デバッグのロード速度が速くなります。また、 <code>nocompress</code> オプションを指定すると、リンク速度を速くすることができます。 本オプションの省略時解釈は、 <code>nocompress</code> です。
備考	<code>form=object,relocate,library,hexadecimal,stype,binary</code> または <code>strip</code> オプションを指定した場合、本オプションは無効です。

シンボル名変更

***RE*Name**

なし

書式	<code>REName = <sub>[,...]</code> <code><sub> : { [<code><ファイル></code>](<code><名前> = <名前>[,...]</code>) </code> <code>[<code><モジュール></code>](<code><名前> = <名前>[,...]</code>) }</code>
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 <code>_</code> 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	<code>rename=(<code>_sym1=</code>data) ;_sym1 を data に変更します。</code> <code>rename=lib1(P=P1) ;ライブラリモジュール lib1 内の P セクションを</code> <code>;P1 セクションに変更します。</code>
備考	<code>extract</code> または <code>strip</code> オプションを指定した場合、本オプションは無効です。

DELete

なし

書 式 DELete = <sub>[,...]
 <sub> : { [<ファイル>](<名前>[,...]) | <モジュール> }

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。
 本オプションでは、シンボル名削除のときオブジェクトは削除しません。

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 オプションを指定した場合、本オプションは無効です。

REPlace

なし

書 式 REPlace = <sub>[,...]
 <sub> : <ファイル名>[(<モジュール名>[,...])]

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き換えます。

例 replace=file1.obj ;モジュール file1 と file1.obj を置換します。
 replace=lib1.lib(md11) ;モジュール md11 とライブラリファイル lib1.lib 内
 ;モジュール md11 を置換します。

備 考 form=object, relocate, absolute, hexadecimal, stype, binary および extract、
 strip オプションを指定した場合、本オプションは無効です。

モジュール抽出

EXtract

なし

書式	EXtract = <モジュール名>[,...]
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 オプションを指定した場合、本オプションは無効です。

デバッグ情報削除

STRip

なし

書式	STRip
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。 本オプションを指定した場合は、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 を指定した場合、本オプションは無効です。

CHange_message

なし

書 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 }`

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。

例 `change_message=warning=2310`
 L2310 をウォーニングレベルに変更し、L2310 出力時も処理を継続します。

`change_message=error`
 全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。

***L*Logo**

***NO*Logo**

なし(常に `nologo` が有効)

書 式 `L`Logo
 `NO`Logo

説 明 コピーライトの出力有無を指定します。
 `logo` オプションを指定した場合、コピーライト表示を出力します。
 `nologo` オプションを指定した場合、コピーライト表示出力を抑制します。
 本オプションの省略時解釈は、`logo` です。

継続指定**END**

なし

書 式 END

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。

終了指定**EXIt**

なし

書 式 EXIt

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： optlnk -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。
本オプション後のコマンドライン指定 nodebug オプションは無効になります。

4.2.8 subcommand file オプション

表 4.14 subcommand カテゴリオプション一覧

項目	コマンドライン形式	ダイアログメニュー	指定内容
1 サブコマンドファイル指定	SUBcommand = <ファイル名>	Subcommand [Use external subcommand file:]	サブコマンドファイルによる オプション指定

サブコマンドファイル指定

Subcommand

Subcommand file[Use external subcommand file:]

書 式 SUBcommand = <ファイル名>

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、サブオプション区切りに空白を指定できます。

サブコマンドファイル内では 1 オプション / 行で指定します。

サブオプションを 1 行に記述できない場合は、&を用いて継続指定できます。

サブコマンドファイル中に本オプションは指定できません。

例 コマンドライン指定 : optlnk 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。

5. 標準ライブラリ構築ツール操作方法

5.1 オプション指定規則

標準ライブラリ構築ツールを起動するコマンドラインの形式は以下の通りです。

```
lbgsh[ <オプション列>... ]  
<オプション列> : -<オプション>[=<サブオプション>[,...]]
```

5.2 オプション解説

標準ライブラリ構築ツールのオプション、サブオプションは、コンパイラオプションに準拠します。以下にコンパイラオプションとの相違を示します。コンパイラオプションの詳細は、「第2章 C/C++コンパイラ操作方法」を参照して下さい。

コマンドライン形式の英大文字は短縮形指定時の文字を、下線は省略時解釈を示します。また、日立統合開発環境の対応するダイアログメニューを、タブ名[項目]で示します。

5.2.1 追加オプション

表 5.1 追加オプション一覧

項目	オプション	ダイアログメニュー	指定内容
1 対象ヘッダ ファイル	Head = <sub>[,...] <sub>:{ ALL RUNTIME CTYPE MATH MATHF STDARG STDIO STDLIB STRING IOS NEW COMPLEX CPPSTRING }	Library Category [Category:]	構築対象ファイルを指定 全てのライブラリ関数 実行時ルーチン ctype.h + 実行時ルーチン math.h + 実行時ルーチン mathf.h + 実行時ルーチン stdarg.h + 実行時ルーチン stdio.h + 実行時ルーチン stdlib.h + 実行時ルーチン string.h + 実行時ルーチン ios + 実行時ルーチン new + 実行時ルーチン complex + 実行時ルーチン string + 実行時ルーチン
2 出力 ファイル	OUTPut = <ファイル名>	Object [Output file:]	出力ライブラリファイル名を指定
3 簡易入出力 関数	NOFLoat	Object [Simple I/O function:]	簡易入出力関数の生成
4 リエントラ ントライブ ラリ	REent	Object [Generate reentrant library:]	リエントラントライブラリを生成

Head

Library Category[Category:]

```
書 式      Head = <sub>[,...]
           <sub>:{ ALL
                   RUNTIME
                   CTYPE
                   MATH
                   MATHF
                   STDARG
                   STDIO
                   STDLIB
                   STRING
                   IOS
                   NEW
                   COMPLEX
                   CPPSTRING }
```

説 明 構築対象ファイルをヘッダファイル名で指定します。
各ヘッダファイルとライブラリ関数との関係は、「10.3 C/C++ライブラリ」を参照してください。実行時ルーチン(runtime)は常に構築対象ファイルになります。
本オプションの省略時解釈は、head=allです。

例 lbgsh -output=sh2.lib -head=mathf -cpu=sh2
mathf.h で定義されたライブラリ関数と実行時ルーチンを-cpu=sh2 でコンパイルし、
ライブラリファイル sh2.lib を出力します。

OUTPut

Object[Output file:]

```
書 式      OUTPut = <ファイル名>
```

説 明 出力ファイル名を指定します。
本オプションの省略時解釈は、output=stdlib.libです。

例 lbgsh -output=sh2.lib -optimize=1 -speed -goptimize -cpu=sh2
全標準ライブラリ用ソースファイルを、-optimize=1 -speed -goptimize -cpu=sh2
でコンパイルし、ライブラリファイル sh2.lib を出力します。

簡易入出力関数

NOFloat

Object[Simple I/O function:]

書式	NOFloat
説明	<p>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。</p> <p><対象関数> fprintf、fscanf、printf、scanf、sprintf、sscanf、vfprintf、vprintf、vsprintf</p>
備考	入出力関数で浮動小数点数を指定した場合に、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。

リエントラントライブラリ

REent

Object[Generate reentrant library:]

書式	REent
説明	リエントラントライブラリを生成します。ただし、rand、srand 関数はリエントラントではありません。
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 文で定義(#define _REENTRANT)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。

5. 標準ライブラリ構築ツール操作方法

5.2.2 指定不可オプション

標準ライブラリ構築ツールで指定できないオプションを表 5.2 に示します。指定した場合は無視されます。

表 5.2 指定不可オプション一覧

項目	オプション	コンパイラ解釈	内容
1 インクルードファイルディレクトリ	include	なし	
2 マクロ名の定義	define	なし	
3 インフォメーションメッセージ	message nomessage	nomessage	出力なし
4 プリプロセッサ展開	preprocessor	なし	
5 オブジェクト形式	code	code = machinecode	機械語プログラムを出力
6 デバッグ情報	debug nodebug	nodebug	出力なし
7 オブジェクトファイル出力指定	objectfile	なし	
8 テンプレートインスタンス生成機能	template	なし	テンプレート機能は使用していません
9 リストファイル	listfile nolistfile	nolistfile	出力なし
10 リスト内容と形式	show	なし	
11 コメントのネスト	comment	なし	コメントネストは使用していません
12 MAC レジスタ保証	macsave	macsave = 1	MACH,MACL レジスタを保証
13 C/C++言語の選択	lang	なし	ファイル拡張子に従います
14 コピーライト出力抑止	logo nologo	nologo	コピーライトの出力を抑止
15 文字列内の文字コード	euc sjis latin1	なし	文字コードは使用していません
16 オブジェクトコード内漢字変換	outcode	なし	文字コードは使用していません

5.2.3 オプション指定時の注意事項

オプション指定時は、次の規則に従ってください。

- (1) cpu、division、endian、fpu、round、denormalize、pic、double=float、rtti オプションはコンパイル時と同じオプションを指定してください。
- (2) #pragma global_register を使用する場合は、preinclude オプションで#pragma global_register 宣言を含むヘッダファイルをインクルード指定してください。日立統合開発環境で指定する場合は、Other[User defined options:]で指定してください。

6. スタック解析ツール操作方法

スタック解析ツールは、最適化リンケージエディタが出力したスタック情報ファイル(*.sni)または日立デバッグインタフェースが出力したプロファイル情報ファイル(*.pro)を読み込んでスタック使用量を表示します。

また、スタック情報ファイルに出力できないアセンブリプログラムのスタック使用量は、編集機能を用いて情報を追加・修正することが可能であり、システム全体のスタック使用量を求めることもできます。

編集したスタック使用量に関する情報は、呼び出し情報ファイル(*.cal)として保存・読み可能です。

6.1 スタック解析ツールの起動

スタック解析ツールを起動するには、Windows®のスタートメニューより“ファイル名を指定して実行”を選択し、Call.exeを指定し実行して下さい。

また、日立統合開発環境をご使用の場合は、Windows®のスタートメニューで“プログラム”を選び、“Hitachi Embedded Workshop”に登録されている“Hitachi Call Walker”を選択して下さい。日立統合環境を起動後は、Toolsメニューより起動することもできます。

6.2 スタック解析ツールの機能概要

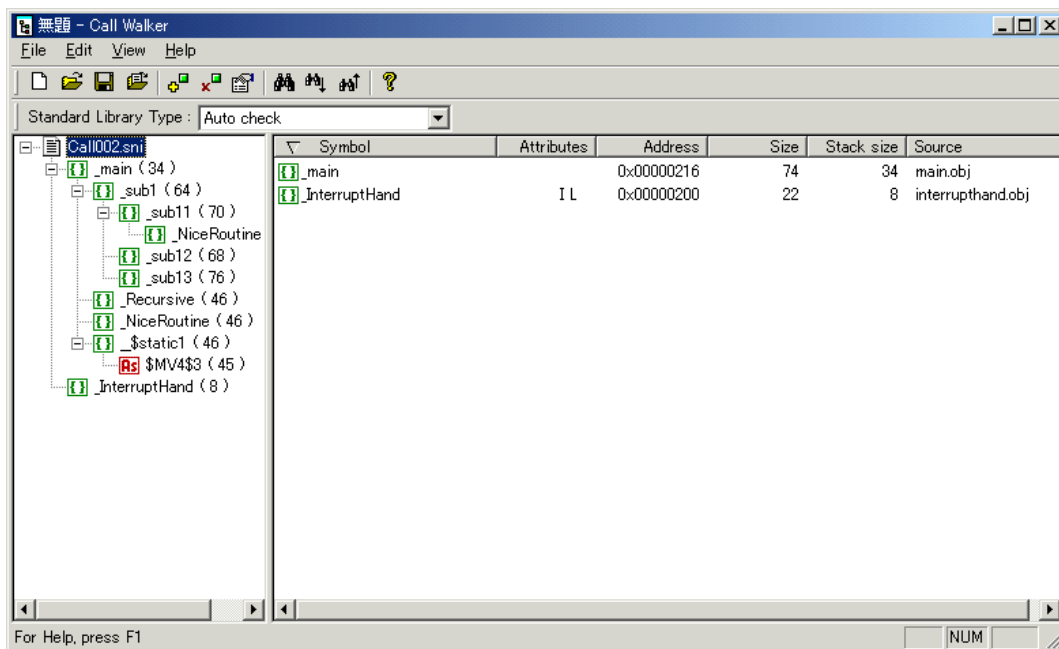


図 6.1 スタック解析ツールの使用例

スタック解析ツールの使用例を図 6.1 に、メニューごとの機能一覧を表 6.1 に示します。左側の呼び出し情報ビューには、関数の呼び出し情報と使用スタック情報が表示されます。右側のシンボル情報ビューには、関数およびシンボルの更に詳しい情報が表示されます。なお、**[C]**はCまたはC++関数を、**[Rs]**はアセンブリ関数を示します。

- 【注】
1. スタック情報ファイルに算出するスタック使用量は、最適化オプション使用時、実際の値より大きくなる場合があります。
 2. 割り込み関数のスタック使用量には、割り込み発生時、割り込みコントローラが退避のために使用するスタックサイズが含まれていません。ご使用のデバイスのハードウェアマニュアルを参照の上、必要量を加算して下さい。

表 6.1 スタック解析ツール機能一覧表

	メニュー	機能
File	New	編集情報をクリアし、新規作成します。
	Open	既存の呼び出し情報ファイル(*.cal)をオープンします。
	Save	現在編集中の呼び出し情報ファイルを上書きで保存します。
	Save As	現在編集中の呼び出し情報ファイルをファイル名を指定して保存します。
	Import Stack File	最適化リンケージエディタが出力するスタック使用量情報ファイル(*.sni)、または日立デバッグインタフェースが出力するプロファイル情報ファイル(*.pro)を読み込みます。
	Recent File	最近使用した呼び出し情報ファイルをオープンします。
	Exit	スタック解析ツールを終了します。
	Edit	Add
Modify		既にある関数およびシンボル情報を編集します。
Delete		関数およびシンボル情報を削除します。
Find		指定した検索条件の関数およびシンボル情報を検索します。
Find Next		Findコマンドにて検索した検索情報で後方検索します。
Find Previous		Findコマンドにて検索した検索情報で前方検索します。
View	Toolbar	ツールバーの表示、非表示を切り替えます。
	Status Bar	ステータスバーの表示、非表示を切り替えます。
	Radix	シンボル情報内の数値部分(Address、Size、Stack size)の基数表示を切り替えます。
Help	Help Topics	スタック解析ツールのヘルプを表示します。
	About Call Walker	スタック解析ツールのバージョンや著作権等を表示します。

さらに詳しい操作方法については、スタック解析ツールのヘルプを参照ください。

7. 環境変数

7.1 環境変数一覧

環境変数の一覧を表 7.1 に示します。

表 7.1 環境変数

環境変数	説明
1 path	実行ファイルの格納ディレクトリを指定します。 指定フォーマット： PC 版 C> path= <実行ファイルパス名>[:<既存パス名>:...] UNIX 版 C シェル %set path =(<実行ファイルパス名> \$path) ポーンシェル %PATH=<実行ファイルパス名>[:<既存パス名>...] %export PATH
2 SHC_LIB	コンパイラのロードモジュールを格納したディレクトリを指定します。PC 版で DOS プロンプトでコンパイルする場合、または UNIX 版の場合はこの環境変数の指定は必須です。 指定フォーマット： PC 版 C> set SHC_LIB= <実行ファイルパス名> UNIX 版 C シェル %setenv SHC_LIB <実行ファイルパス名> ポーンシェル %SHC_LIB = <実行ファイルパス名> %export SHC_LIB
3 SHCPU	コンパイラ・アセンブラの cpu オプションによる CPU 種別の指定を、環境変数によって指定します。 <CPU> SH1 SH2 SH2E SHDSP SH3 SH3DSP SH4 SHCPU 環境変数による CPU の指定と、cpu オプションによる CPU の指定が異なる場合は、ウォーニングメッセージを出力し cpu オプションの指定を優先します。 コンパイラでは SHDSP、SH3DSP を指定するとそれぞれ SH2、SH3 を指定したと解釈します。 指定フォーマット： PC 版 C> set SHCPU= <CPU> UNIX 版 C シェル %setenv SHCPU <CPU> ポーンシェル %SHCPU=<CPU> %export SHCPU

7. 環境変数

環境変数	説明
4 SHC_INC	<p>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。</p> <p>システムインクルードファイルの検索順序は、include オプション指定ディレクトリ、SHC_INC 指定ディレクトリ、システムディレクトリ(SHC_LIB)となります。</p> <p>ユーザインクルードファイルの検索順序は、カレントディレクトリ、include オプション指定ディレクトリ、SHC_INC 指定ディレクトリ、システムディレクトリ(SHC_LIB)となります。</p> <p>この環境変数の指定がない場合、UNIX 版では SHC_LIB を仮定します。PC 版には省略時解釈はありません。</p> <p>指定フォーマット：</p> <p>PC 版 C> set SHC_INC = <インクルードパス名> [<インクルードパス名> ;...]</p> <p>UNIX 版 C シェル</p> <pre>%setenv SHC_INC <インクルードパス名>[:<インクルードパス名> :...]</pre> <p>ポーンシェル</p> <pre>%SHC_INC = <インクルードパス名>[:<インクルードパス名> :...]</pre> <pre>%export SHC_INC</pre>
5 SHC_TMP	<p>コンパイラが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。PC 版で、DOS プロンプトでコンパイルする場合は、この環境変数の指定は必須です。UNIX 版では、この環境変数の指定がない場合、環境変数 TMPDIR が指定されていれば TMPDIR が示すディレクトリを、SHC_TMP、TMPDIR が指定されていなければ、/usr/tmp にテンポラリファイルを作成します。</p> <p>指定フォーマット：</p> <p>PC 版 C> set SHC_TMP= <テンポラリファイルパス名></p> <p>UNIX 版 C シェル %setenv SHC_TMP <テンポラリファイルパス名></p> <p>ポーンシェル %SHC_TMP = <テンポラリファイルパス名></p> <pre>%export SHC_TMP</pre>
6 HLNK_LIBRARY1 HLNK_LIBRARY2 HLNK_LIBRARY3	<p>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。</p> <p>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。</p> <p>指定フォーマット：</p> <p>PC 版</p> <pre>C> set HLNK_LIBRARY1= <ライブラリ名 1> C> set HLNK_LIBRARY2= <ライブラリ名 2> C> set HLNK_LIBRARY3= <ライブラリ名 3></pre> <p>UNIX 版 C シェル</p> <pre>%setenv HLNK_LIBRARY1 <ライブラリ名 1> %setenv HLNK_LIBRARY2 <ライブラリ名 2> %setenv HLNK_LIBRARY3 <ライブラリ名 3></pre> <p>ポーンシェル %HLNK_LIBRARY1=<ライブラリ名 1></p> <pre>%export HLNK_LIBRARY1 %HLNK_LIBRARY2=<ライブラリ名 2> %export HLNK_LIBRARY2 %HLNK_LIBRARY3=<ライブラリ名 3> %export HLNK_LIBRARY3</pre>

環境変数	説明
7 HLNK_TMP	<p>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ディレクトリを指定します。この環境変数の指定がない場合は、カレントディレクトリにテンポラリファイルを作成します。</p> <p>指定フォーマット：</p> <p>PC 版 C> set HLNK_TMP= <テンポラリファイルパス名></p> <p>UNIX 版 C シェル %setenv HLNK_TMP <テンポラリファイルパス名></p> <p> ポーンシェル%HLNK_TMP = <テンポラリファイルパス名></p> <p> %export HLNK_TMP</p>
8 HLNK_DIR	<p>最適化リンケージエディタの入力ファイル格納ディレクトリを指定します。</p> <p>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、カレントディレクトリ、HLNK_DIR 指定ディレクトリになります。</p> <p>ただし、ワイルドカードで指定したファイルは、カレントディレクトリ内だけ検索します。</p> <p>指定フォーマット：</p> <p>PC 版 C> set HLNK_DIR= <入力ファイルパス名>[;<入力ファイルパス名>;...]</p> <p>UNIX 版 C シェル</p> <p> %setenv HLNK_DIR <入力ファイルパス名>[:<入力ファイルパス名> :...]</p> <p> ポーンシェル</p> <p> %HLNK_DIR = <入力ファイルパス名> [:<入力ファイルパス名> :...]</p> <p> %export HLNK_DIR</p>

【注】* 複数ディレクトリを指定する場合は、PC 版は";"(セミコロン)または","(カンマ)で、UNIX 版は":"(コロン)で区切ってください。

7.2 コンパイラの暗黙の宣言

コンパイラについては、オプション指定やバージョンに合わせて、以下のような暗黙の#define宣言が行われます。

表 7.2 暗黙の宣言

	オプション	暗黙の宣言
1	cpu=sh1	#define _SH1
	cpu=sh2	#define _SH2
	cpu=sh2e	#define _SH2E
	cpu=sh3	#define _SH3
	cpu=sh4	#define _SH4
2	pic=1	#define _PIC
3	endian=big	#define _BIG
4	endian=little	#define _LIT
5	double=float	#define _FLT, #define __FLT__
6	fpu=single	#define _FPS
7	fpu=double	#define _FPD
8	denormalize=on	#define _DON
9	round=nearest	#define _RON
10	-	#define __HITACHI_VERSION__ ^{*1}
11	-	#define __HITACHI__ ^{*2}

【注】*1 __HITACHI_VERSION__の値は、次のように参照します。

- ・C ソースプログラム内：__HITACHI_VERSION__=aabb
aa: version 部分
bb: revision 部分
- ・コンパイラ内での定義例

```
#define __HITACHI_VERSION__ 0x0501 // Version 5.1C の場合
```

```
#define __HITACHI_VERSION__ 0x0700 // Version 7.0 の場合
```

*2 常に定義されます

8. ファイル仕様

8.1 ファイル名の付け方

ファイル名指定時に拡張子を省略した場合、標準のファイル拡張子を付加したファイル名を使用します。日立開発環境で使用する標準のファイル拡張子を表 8.1 に示します。

表 8.1 日立開発環境で使用する標準のファイル拡張子

No.	拡張子	意味
1	c	C ソースプログラムファイル
2	cpp,cc,cp	C++ソースプログラムファイル
3	h	インクルードファイル
4	lis,lst ¹	C プログラム用リストファイル
5	lis,lpp ¹	C++プログラム用リストファイル
6	p	C プログラム用プリプロセッサ展開ファイル
7	pp	C++プログラム用プリプロセッサ展開ファイル
8	src	アセンブリソースプログラムファイル
9	exp	アセンブリプログラム用プリプロセッサ展開ファイル
10	lis	アセンブリプログラム用リストファイル
11	obj	リロケータブルオブジェクトプログラムファイル
12	rel	リロケータブルロードモジュールファイル
13	abs	アブソリュートロードモジュールファイル
14	map	リンケージリストファイル
15	lib	ライブラリファイル
16	lbp	ライブラリリストファイル
17	mot	S タイプフォーマット
18	hex	HEX フォーマット
19	bin	バイナリファイル
20	fsy	最適化リンケージエディタ出力シンボルアドレスファイル
21	sni	スタック情報ファイル
22	pro	プロファイル情報ファイル
23	dbg	DWARF2 フォーマットデバッグ情報ファイル
24	rti	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
25	cal	呼び出し情報ファイル
26	bis	外部シンボル割り付け情報ファイル

【注】 *1 UNIX 版では lis、PC 版では lst または lpp です。

rti_ で始まるファイル名はシステム予約名ですので使用しないでください。
各プロジェクトで生成される tpldir のフォルダの下に出力されるファイル拡張子を表 8.2 に示します。

表 8.2 tpldir フォルダ出力ファイル

No.	拡張子	意味
1	td	tentative 定義の変数情報
2	ti	テンプレート情報ファイル
3	pl	パラメタ情報ファイル
4	ii	インスタンス情報ファイル

ファイル名の付け方の一般的な規則は、各ホストマシンに準じています。ご使用になるホストマシンのマニュアルを参照してください。

8.2 コンパイルリストの参照方法

本節では、コンパイルリストの内容と形式について説明します。

8.2.1 コンパイルリストの構成

コンパイルリストの構成と内容を表 8.3 に示します。

表 8.3 コンパイル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 ^{*1}	オプション省略時
1	ソースリスト情報	ソースプログラムのリスト ^{*2}	show=source show=nosource	出力しない
		インクルードファイル展開後のソースプログラムのリスト ^{*3}	show=include show=noinclude	出力しない
		マクロ展開後のソースプログラムのリスト ^{*3}	show=expansion show=noexpansion	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、アセンブリコード	show=object show=noobject	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、シンボル数	show=statistics show=nostatistics	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		出力する

【注】 *1 すべてのオプションは listfile オプションを指定した場合に有効です。

*2 ソースプログラムのリストは show=object を同時に指定した場合、オブジェクト情報内に出力されます。

*3 インクルードファイル、マクロ展開後のソースプログラムのリストは show=source を指定した場合に有効です。

8.2.2 ソースリスト情報

ソースリスト情報の出力形式には、プリプロセッサを通す前のソースプログラムを出力する形式 (show=noinclude,noexpansion を指定する場合)と、プリプロセッサを通した後のソースプログラムを出力する形式 (show=include,expansion を指定する場合)があります。それぞれの出力形式を図 8.1、図 8.2 に示します。また、図 8.2 に相違点を網掛けで示します。

```

***** SOURCE LISTING *****
FILE NAME: m0260.c

Seq      File      Line  0-----1-----2-----3-----4-----5---
  1      m0260.c    1      #include "header.h"
  4      m0260.c    2
  5      m0260.c    3      int sum2(void)
  6      m0260.c    4      {   int j;
  7      m0260.c    5
  8      m0260.c    6      #ifdef SMALL
  9      m0260.c    7          j=SML_INT;
 10      m0260.c    8      #else
 11      m0260.c    9          j=LRG_INT;
 12      m0260.c   10      #endif
 13      m0260.c   11
 14      m0260.c   12          return j; /* continue123456789012345678901234567
(1)      (2)      (3)      ±2345678901234567890 */
 15      m0260.c   13      (7)  }

```

図 8.1 show=noinclude,noexpansion のソースリスト情報

```

***** SOURCE LISTING *****
FILE NAME: m0260.c

Seq      File      Line  0-----1-----2-----3-----4-----5---
  1      m0260.c    1      #include "header.h"
  2      header.h   1      #define SML_INT      1 (4)
  3      header.h   2      #define LRG_INT     100
  4      m0260.c    2
  5      m0260.c    3      int sum2(void)
  6      m0260.c    4      {   int j;
  7      m0260.c    5
  8      m0260.c    6      #ifdef SMALL
  9      m0260.c    7      X   j=SML_INT;
 10      m0260.c    8      (5) #else
 11      m0260.c    9      E   j=100;
 12      m0260.c   10      (6) #endif
 13      m0260.c   11
 14      m0260.c   12          return j; /* continue123456789012345678901234567
(1)      (2)      (3)      ±2345678901234567890 */
 15      m0260.c   13      (7)  }

```

図 8.2 show=include,expansion のソースリスト情報

- (1) リスト上の行番号
- (2) ソースプログラムファイル名またはインクルードファイル名
- (3) ソースプログラムまたはインクルードファイル内の行番号
- (4) show = include 指定時、インクルードファイルの展開のあったソース行
- (5) show = expansion 指定時、#ifdef 文、#elif 文等の条件コンパイル文でコンパイル対象と
ならないソース行
- (6) show = expansion 指定時、#define 文によるマクロ置換のあったソース行
- (7) ソースプログラムの 1 行がコンパイルリストの 1 行に入りきらず、複数行にまたがって
表示されたソース行

8.2.3 オブジェクト情報

オブジェクト情報の出力形式には、ソースプログラムを出力する形式(show=source, object を指定する場合)とソースプログラムを出力しない形式(show=nosource,object を指定する場合)があります。それぞれのリスト例を図 8.3 および図 8.4 に示します。

```

***** OBJECT LISTING *****
FILE NAME: m0251.c

SCT   OFFSEI  CODE      C LABEL  INSTRUCTION OPERAND      COMMENT
(1)   (2)     (3)
      m0251.c  1    extern int multipli(int);
      m0251.c  2
      m0251.c  3    int multipli(int x)
P
00000000          _multipli:                                ; function: multipli
                                                ; frame size=16 (7)
00000000  4F22          STS.L   PR,R15
00000002  7FF4          ADD    #-12,R15
00000004  1F42          MOV.L  R4,@(8,R15)
      m0251.c  4    {
      m0251.c  5          int i;
      m0251.c  6          int j;
      m0251.c  7
      m0251.c  8          j=1;
00000006  E201          MOV    #1,R2
00000008  2F22          MOV.L  R2,@R15
      m0251.c  9          for(i=1; i<=x; i++){
0000000A  E301          MOV    #1,R3
0000000C  1F31          MOV.L  R3,@(4,R15)
0000000E  A009          BRA    L213
00000010  0009          NOP
00000012          L214:
      m0251.c 10          j*=i;
00000012  50F1          MOV.L  @(4,R15),R0
00000014  61F2          MOV    @R15,R1
00000016  D30A          MOV.L  L216+2,R3          ; __multi
00000018  430B          JSR    @R3
      :
      .

```

図 8.3 show=source,object のオブジェクト情報

8. ファイル仕様

```

***** OBJECT LISTING *****
FILE NAME: m0251.c

  SCT   OFFSET   CODE   C LABEL   INSTRUCTION OPERAND   COMMENT
  (1)   (2)     (3)                   (4)                   (5)
P
      00000000   ; File m0251.c , Line 3   ; block
      _multipli: (6)   ; function: multipli
                        ; frame size=16 (7)
      00000000   4F22   STS.L   PR,@-R15
      00000002   7FF4   ADD    #-12,R15
      00000004   1F42   MOV.L  R4,@(8,R15)
                        ; File m0251.c , Line 4   ; block
                        ; File m0251.c , Line 8   ; expression statement
      00000006   E201   MOV    #1,R2
      00000008   2F22   MOV.L  R2,@R15
                        ; File m0251.c , Line 9   ; for
      0000000A   E301   MOV    #1,R3
      0000000C   1F31   MOV.L  R3,@(4,R15)
      0000000E   A009   BRA    L213
      00000010   0009   NOP
      00000012   L214:
                        ; File m0251.c , Line 9   ; block
                        ; File m0251.c , Line 10  ; expression statement
      00000012   50F1   MOV.L  @(4,R15),R0
      00000014   61F2   MOV.L  @R15,R1
      00000016   D30A   MOV.L  L216+2,R3   ; __multi
      00000018   430B   JSR    @R3
      .
      .
      .
  
```

図 8.4 show=nosource,object のオブジェクト情報

- (1) 各セクションのセクション名(P、C、D、B、C\$INIT、C\$VTBL)
- (2) 各セクションの先頭からのオフセット
- (3) 各セクションのオフセットアドレスの内容
- (4) 機械語に対応するアセンブリコード
- (5) プログラムに対応するコメント(非最適化時のみ出力、ラベルは最適化時も出力)
- (6) プログラムの行情報(非最適化時のみ出力)
- (7) スタックフレームサイズ(バイト数)

8.2.4 統計情報

統計情報の出力例を図 8.5 に示します。

```

***** STATISTICS INFORMATION *****

***** ERROR INFORMATION ***** (1)

NUMBER OF ERRORS:          0
NUMBER OF WARNINGS:        0
NUMBER OF INFORMATIONS:    0

***** SOURCE LINE INFORMATION ***** (2)

COMPILED SOURCE LINE:      13

***** SECTION SIZE INFORMATION ***** (3)

PROGRAM      SECTION(P):          0x000044 Byte(s)
CONSTANT     SECTION(C):          0x000000 Byte(s)
DATA         SECTION(D):          0x000000 Byte(s)
BSS          SECTION(B):          0x000000 Byte(s)

TOTAL PROGRAM SECTION: 00000044 Byte(s)
TOTAL CONSTANT SECTION: 00000000 Byte(s)
TOTAL DATA   SECTION: 00000000 Byte(s)
TOTAL BSS     SECTION: 00000000 Byte(s)

TOTAL PROGRAM SIZE: 0x000044 Byte(s)

***** LABEL INFORMATION ***** (4)

NUMBER OF EXTERNAL REFERENCE SYMBOLS: 1
NUMBER OF EXTERNAL DEFINITION SYMBOLS: 1
NUMBER OF INTERNAL/EXTERNAL SYMBOLS: 6

```

図 8.5 統計情報

- (1) レベル別メッセージの総数
- (2) ソースファイルのコンパイルした行数
- (3) 各セクションのサイズとその合計
- (4) オブジェクトプログラムの外部参照シンボルの数、外部定義シンボルの数、内部ラベルと外部ラベルの合計数

【注】 message オプションが指定されていない場合には、レベル別メッセージ(1)の NUMBER OF INFORMATIONS は出力されません。noobject オプションを指定した場合とエラーレベル、フェータルレベルのエラーが発生した場合には、セクションサイズ情報(3)とラベル情報(4)は出力されません。また、code=asmcode を指定した場合は、セクションサイズ情報(3)は当該セクションの有無を 0 と 1 で示すようになります。

8.2.5 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。コマンド指定情報の出力例を図 8.6 に示します。

```
*** COMMAND PARAMETER ***  
-listfile test.c
```

図 8.6 コマンド指定情報

8.3 アセンブルリストの参照方法

8.3.1 アセンブルリストの構成

アセンブルリストの構成と内容を表 8.4 に示します。

表 8.4 アセンブルリストの構成と内容

No	リストの作成	内容	オプション ^{*1}	オプション省略時
1	ソースリスト情報	ソースプログラムに関する情報を示します。	source	出力する
2	クロスリファレンス リスト情報	ソースプログラムのシンボルに関する情報を示します。	cross_reference	出力する
3	セクション情報リスト	ソースプログラムのセクションに関する情報を示します。	section	出力する

【注】 *1 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。

8.3.2 ソースリスト情報

ソースリスト情報を出力します。ソースリスト情報の出力例を図 8.7 に示します。

```

PROGRAM NAME =                "SAMPLE"                (7)
 1                               1                .HEADING "" "SAMPLE" ""
 2                               2 POINT          .ASSIGNA 16
 3                               3 Parm1         .REG (R0)
 4                               4 Parm2         .REG (R1)
 5                               5 WORK1         .REG (R2)
 6                               6 WORK2         .REG (R3)
 7                               7 WORK3         .REG (R4)
 8                               8 WORK4         .REG (R5)
~
20 00000000                    9 I1 FIX_MUL :
21 00000000 2107                10 I1      DIV0S Parm1, Parm2
22 00000002 0229                11 I1      MOV  WORK1
23 00000004 4011                12 I1      CMP/PZ Parm1
24 00000006 8900                13 I1      BT   MUL01
25 00000008 600B                14 I1      NEG  Parm1, Parm1
(1) (2)      (3)      (4) (5)                                (6)
~
231                               ***** BEGIN-POOL *****
232 00000180 00018000           DATA FOR SOURCE-LINE 17
233 00000184 00024000           DATA FOR SOURCE-LINE 18
234 00000188 00030000           DATA FOR SOURCE-LINE 19
235 0000018C 00050000           DATA FOR SOURCE-LINE 20
236                               ***** END-POOL *****
237                               35                .END
****TOTAL ERRORS                0
****TOTAL WARNINGS              0
(9)

```

図 8.7 ソースプログラムリスト

- (1) 行番号(10進)です。
- (2) ロケーションカウンタ値(16進)です。
- (3) オブジェクトコード(16進)です。オペレーションが.RES、.SRES、.SRESC、.SRESZ、.FRESのいずれかであるときは、確保する領域のサイズをバイト単位で示します。
- (4) ソース行番号(10進)です。
- (5) 展開区分
ファイルインクルード、条件つきアセンブリ機能、マクロ機能が展開した区分を示します。
In ファイルインクルード(nはインクルードのネストレベルを示します)
C 条件つきアセンブルの成立、繰り返し展開、条件付き繰り返し展開
M マクロ展開
- (6) ソースステートメントです。
- (7) .HEADINGで設定したヘッダです。
- (8) リテラルプールです。
- (9) エラーとウォーニングの総数です。エラーメッセージは、エラーが存在するソースステートメントの次の行に出力されます。

8.3.3 クロスリファレンスリスト

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図 8.8 に示します。

*** CROSS REFERENCE LIST						
NAME	SECTION	ATTR	VALUE	SEQUENCE		
FIX_DIV	SAMPLE		00000088	91*	223	
FIX_MUL	SAMPLE		00000000	19*	218	
MUL01	SAMPLE		0000000A	23	25*	
MUL02	SAMPLE		00000010	26	28*	
MUL03	SAMPLE		00000082	87	89*	
Parm1		REG		3*	20	22 24 24
					28	29 29 31 32
					32	35 36 36 38
					40	45 49 55 57
					59	61 63 65 67
					69	71 73 75 77
					79	81 83 85 88
					88	93 94 99 101
Parm2		REG		4*	20	25 27 27
					28	31 33 33 35
					38	41 43 44 46
					48	54 56 58 60
					62	64 66 68 70
(1)	(2)	(3)	(4)	(5)		

図 8.8 クロスリファレンスリスト

8. ファイル仕様

- (1) シンボルの名称です。
- (2) シンボルが含まれるセクションの名称です(セクション名の最初の8文字)。
- (3) シンボルの属性です。
EXPT 外部定義シンボル
IMPT 外部参照シンボル
SCT セクション名
REGREG アセンブラ制御命令で定義したシンボル
FREGFREG アセンブラ制御命令で定義したシンボル
ASGNASSIGN アセンブラ制御命令で定義したシンボル
EQUEQU アセンブラ制御命令で定義したシンボル
MDEF 二重定義したシンボル
UDEF 未定義シンボル
表示なし..... その他のシンボル
- (4) シンボルの値です(16進)。
- (5) シンボルを定義、参照しているリスト行番号(10進)です。
行番号の後ろのアスタリスク(*)は、定義行であることを示します。

8.3.4 セクション情報リスト

セクション情報を出力します。セクション情報の出力例を図 8.9 に示します。

*** SECTION DATA LIST			
SECTION	ATTRIBUTE	SIZE	START
SAMPLE	REL-CODE	000000190	
(1)	(2)	(3)	(4)

図 8.9 セクション情報リスト

- (1) セクションの名称です。
- (2) セクションの種類です。
REL 相対アドレスセクション
ABS 絶対アドレスセクション
CODE コードセクション
DATA データセクション
STACK スタックセクション
DUMMY ダミーセクション
- (3) セクションのサイズです(16進数、単位はバイト)。
- (4) 絶対アドレスセクションの先頭アドレスです。

8.4 リンケージリストの参照方法

最適化リンケージエディタが出力するリンケージリストの内容と形式について説明します。

8.4.1 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 8.5 に示します。

表 8.5 リンケージ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 ^{*1}	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	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		出力する
4	シンボル情報	静的定義シンボル名、アドレス、サイズ、種別をアドレス順に表示	show=symbol	出力しない
		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=reference	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	変数アクセス最適化対象シンボル情報	8bit/16bit 絶対アドレッシングモードでの参照回数を表示	show=reference	出力しない
7	関数アクセス最適化対象シンボル情報	シンボルの参照回数を表示	show=reference	出力しない

【注】 *1 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。

8.4.2 オプション情報

コマンドライン、サブコマンドファイルで指定したオプション列を出力します。オプション情報の出力例を図 8.10 に示します(optlnk -sub=test.sub -list -show 指定時)。

```
(test.subの内容)
INPUT test.obj

*** Options ***
-sub=test.sub
INPUT test.obj (2) } (1)
-list
-show
```

図 8.10 オプション情報の出力例(リンケージリスト)

- (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。

8.4.3 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図 8.11 に示します。

```
*** Error information ***  
  
** L2310 (E) Undefined external symbol "strcmp" referred to in "test.obj" } (1)
```

図 8.11 エラー情報の出力例 (リンケージリスト)

- (1) エラーメッセージを出力します。

8.4.4 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図 8.12 に示します。

```
*** Mapping List ***  
  
SECTION           START      END         SIZE   ALIGN  
(1)              (2)         (3)         (4)     (5)  
P  
C  
D  
B  
00000000  000004d6    4d6    2  
000004d6  00000533    5d     2  
00000534  0000053c    8      2  
0000053c  00004112   3bd6   2
```

図 8.12 リンケージマップ情報の出力例(リンケージリスト)

- (1) セクション名を表示します。
- (2) 先頭アドレスを表示します。
- (3) 最終アドレスを表示します。
- (4) セクションサイズを表示します。
- (5) セクションの境界調整数を表示します。

8.4.5 シンボル情報

show=symbol を指定した場合、外部定義シンボルまたは静的内部定義シンボルのアドレス、サイズ、種別をアドレス順に出力します。また、show=reference を指定した場合は、各シンボルの参照回数、最適化実行の有無も出力します。シンボル情報の出力例を図 8.13 に示します。

```

*** Symbol List ***

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

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

```

図 8.13 シンボル情報の出力例(リンケージリスト)

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

8.4.6 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図 8.14 に示します。

```
*** Delete Symbols ***

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

図 8.14 シンボル削除情報の出力例(リンケージリスト)

- (1) 削除シンボル名を表示します。
- (2) 削除シンボルサイズを表示します。
- (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
```

8.4.7 変数アクセス最適化対象シンボル情報

SuperH RISC engine マイコン向けでは本情報は出力されません(図 8.15)。

```
*** Variable Accessible with Abs8 ***

SYMBOL                                SIZE    COUNTS  OPTIMIZE

*** Variable Accessible with Abs16 ***

SYMBOL                                SIZE    COUNTS  OPTIMIZE
```

図 8.15 変数アクセス最適化対象シンボル情報の出力例(リンケージリスト)

8.4.8 関数アクセス最適化対象シンボル情報

SuperH RISC engine マイコン向けでは本情報は出力されません(図 8.16)。

```
*** Function Call ***

SYMBOL                                COUNTS  OPTIMIZE
```

図 8.16 関数アクセス最適化対象シンボル情報の出力例(リンケージリスト)

8.5 ライブラリリストの参照方法

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。

8.5.1 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表 8.6 に示します。

表 8.6 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 ¹	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】 *1 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。

8.5.2 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図 8.17 に示します(optlnk -sub=test.sub -list -show を指定した場合)。

```
(test.sub の内容)

form library
in adhry.obj
output test.lib

*** Options ***

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

} (2) } (1)

図 8.17 オプション情報の出力例(ライブラリリスト)

- (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。

8.5.3 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図 8.18 に示します。

```
*** Error information ***  
** L1200 (W) Backed up file "main.lib" into "main.lbk" } (1)
```

図 8.18 エラー情報の出力例(ライブラリリスト)

- (1) エラーメッセージを出力します。

8.5.4 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図 8.19 に示します。

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

図 8.19 ライブラリ情報の出力例(ライブラリリスト)

- (1) ライブラリ名を表示します。
- (2) CPU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。

8.5.5 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図 8.20 に示します。

```

*** Library List ***

MODULE      LAST UPDATE
(1)         (2)
SECTION
(3)
SYMBOL
(4)
adhry
                29-Feb-2000 12:34:56

P
  _main
  _Proc0
  _Procl

C

D
  _Version

B
  _IntGlob
  _CharGlob

```

図 8.20 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(ライブラリリスト)

- (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。

9. プログラミング

9.1 プログラムの構造

9.1.1 セクション

コンパイラ、アセンブラが出力するオブジェクトプログラムの実行命令、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 データを格納します。
stack スタック領域です。
- 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。同一セクション内で初期値があるデータと初期値がないデータは混在できません。一つでも初期値があると、初期値のない領域は0で初期化します。
- 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。
- 境界調整数
セクションを割り付けるアドレスの補正值です。最適化リンケージエディタでは、境界調整数の倍数アドレスになるよう、アドレスを補正します。

9.1.2 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 9.1 に示します。

表 9.1 メモリ領域の種類とその性質の概要

名称	セクション		形式種別	初期値 書き込み操作	境界調整数	内容
	名称	属性				
1 プログラム領域	P ¹	code	相対形式	有 不可	4byte ²	機械語を格納
2 定数領域	C ¹	data	相対形式	有 不可	4byte	const 型のデータを格納
3 初期化データ領域	D ¹	data	相対形式	有 可	4byte	初期値のあるデータを格納
4 未初期化データ領域	B ¹	data	相対形式	無 可	4byte	初期値のないデータを格納

9. プログラミング

名称	セクション		形式 種別	初期値 書き込 み操作	境界 調整数	内容
	名称	属性				
5 GBR セクション	\$G0	data	相対 形式	有 可	4byte	#pragma gbr_base で指定された初期値のあるデータを格納。初期値のない場合は0を格納。
6 GBR セクション	\$G1	data	相対 形式	有 可	4byte	#pragma gbr_base1 で指定された初期値のあるデータを格納。初期値のない場合は0を格納。
7 C++初期処理 / 後処理データ領域	C\$INIT	data	相対 形式	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
8 C++仮想関数表領域	C\$VTBL	data	相対 形式	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
9 スタック領域	S	stack	相対 形式	無 可	4byte	プログラム実行に必要な領域。「9.2.1(2) 動的領域の割り付け」参照。
10 ヒープ領域			相対 形式	無 可		ライブラリ関数 malloc、realloc、calloc、new で使用する領域。「9.2.1(2) 動的領域の割り付け」参照。

- 【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。
*2 align16 オプションを指定している場合、16byte になります。

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。

```
int a=1;
char b;
const int c=0;
void main(){
    ...
}
```

Cプログラム

プログラム領域(main(){...})	P
定数領域(c)	C
初期化データ領域(a)	D
未初期化データ領域(b)	B

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

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。

```
class A{
  int m;
  A(int p);
  ~A();
};
A a(1);
int b;
extern const char c='a';
int d=1;
void f(){...}
```

C++プログラム

プログラム領域 (f(){...})	P
定数領域 (c)	C
初期化データ領域 (d)	D
未初期化データ領域 (a,b)	B
初期処理/後処理データ領域 (&A::A、&A::~A)	C\$INIT

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

9.1.3 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 を用いてセクションの開始や属性、形式種別を宣言します。
.SECTION の宣言形式は次のとおりです。詳細は「11.4 アセンブラ制御命令」を参照してください。

```
.SECTION <セクション名>[,<セクション属性>[,<形式種別>]]  
  <形式種別>: 相対アドレス形式セクションの場合、ALIGN=<境界調整数>  
             絶対アドレス形式セクションの場合、LOCATE=<アドレス値>
```

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。

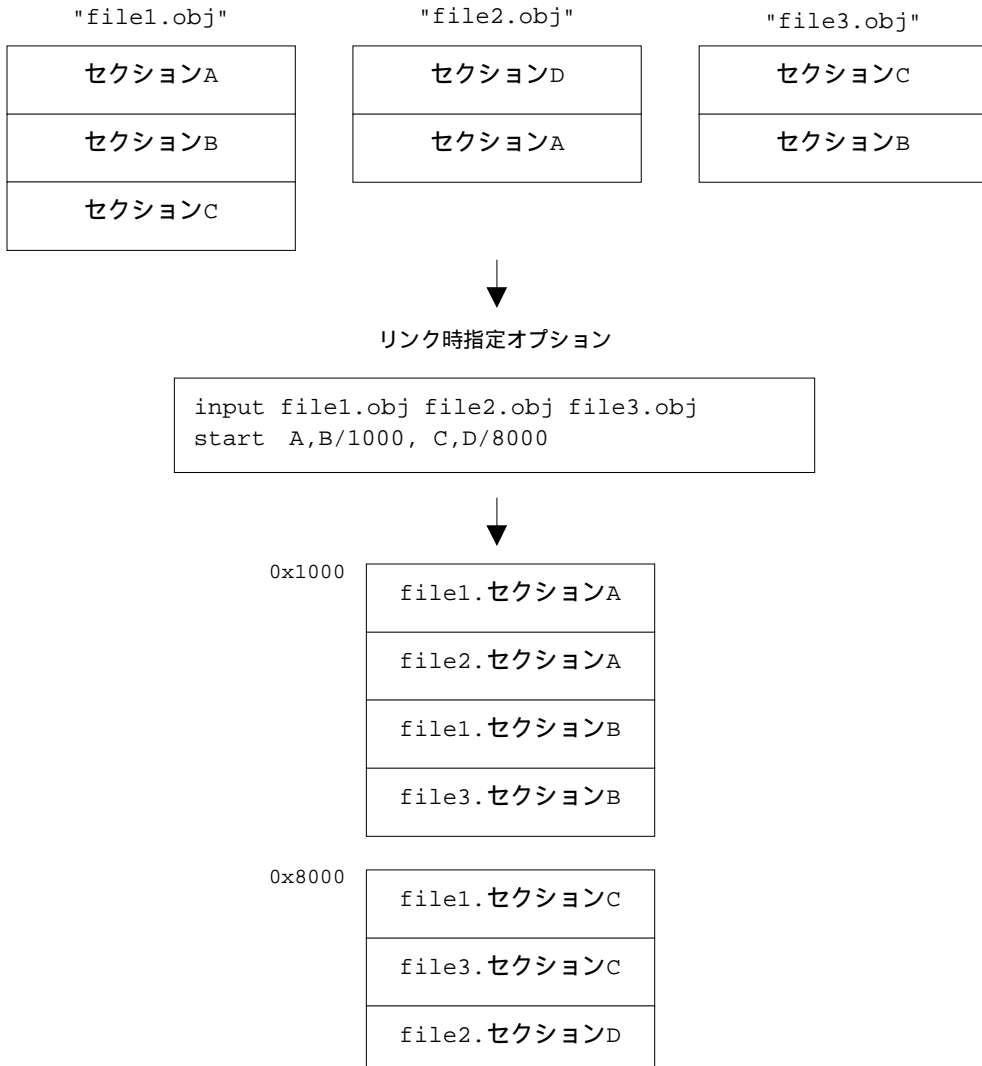
```
.CPU      SH2  
.OUTPUT   DBG  
SIZE:     .EQU      8  
  
          .SECTION  A, CODE, ALIGN=4           ; (1)  
START:  
          MOV.L     LITERAL, R0  
          MOV.L     LITERAL+4, R1  
          MOV.L     #SIZE, R2  
LOOP:  
          CMP.PL    R2  
          BF        EXIT  
          MOV.B     @R0+, R3  
          MOV.B     R3, @R1  
          ADD       #-1, R2  
          ADD       #1, R1  
          BRA       LOOP  
          NOP  
EXIT:  
          SLEEP  
          NOP  
LITERAL:  
          .DATA.L   CONST  
          .DATA.L   DATA  
;  
          .SECTION  B, DATA, LOCATE=H'00002000 ; (2)  
CONST:  
          .DATA.B   H'01, H'02, H'03, H'04, H'05, H'06, H'07, H'08  
;  
          .SECTION  C, STACK, ALIGN=4         ; (3)  
DATA:  
          .RES.B    8  
          .END
```

- (1) セクション名 A、境界調整数 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 H'2000、絶対アドレス形式の data セクションを宣言しています。
- (3) セクション名 C、境界調整数 4、相対アドレス形式の stack セクションを宣言しています。

9.1.4 セクションの結合

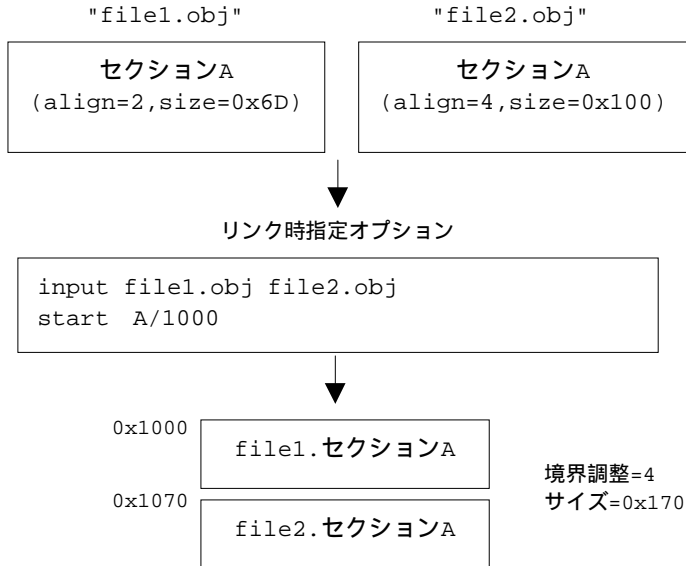
最適化リンケージエディタでは、入力オブジェクトプログラム内の同一セクションを結合し、start オプションによって指定されたアドレスに割り付けます。

(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。

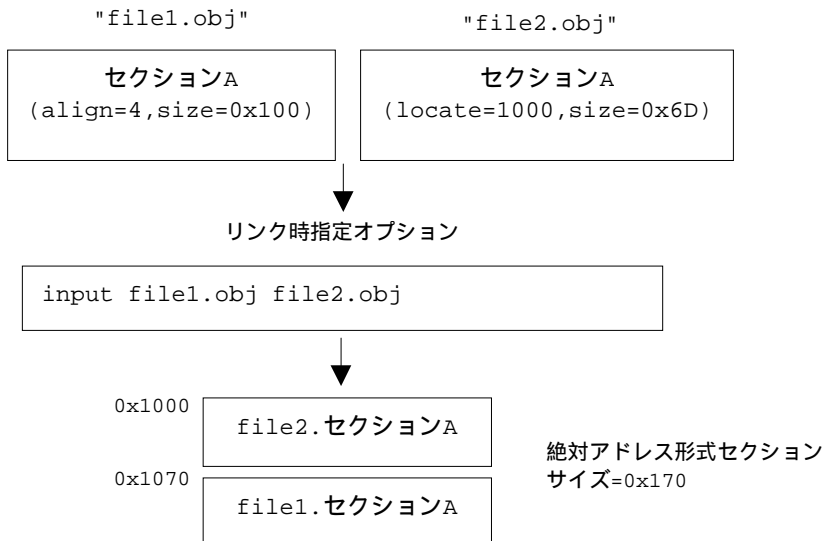


9. プログラミング

(2) 境界調整数の異なる同名セクションは、境界調整後に結合します。セクションの境界調整数は大きい方に合わせます。

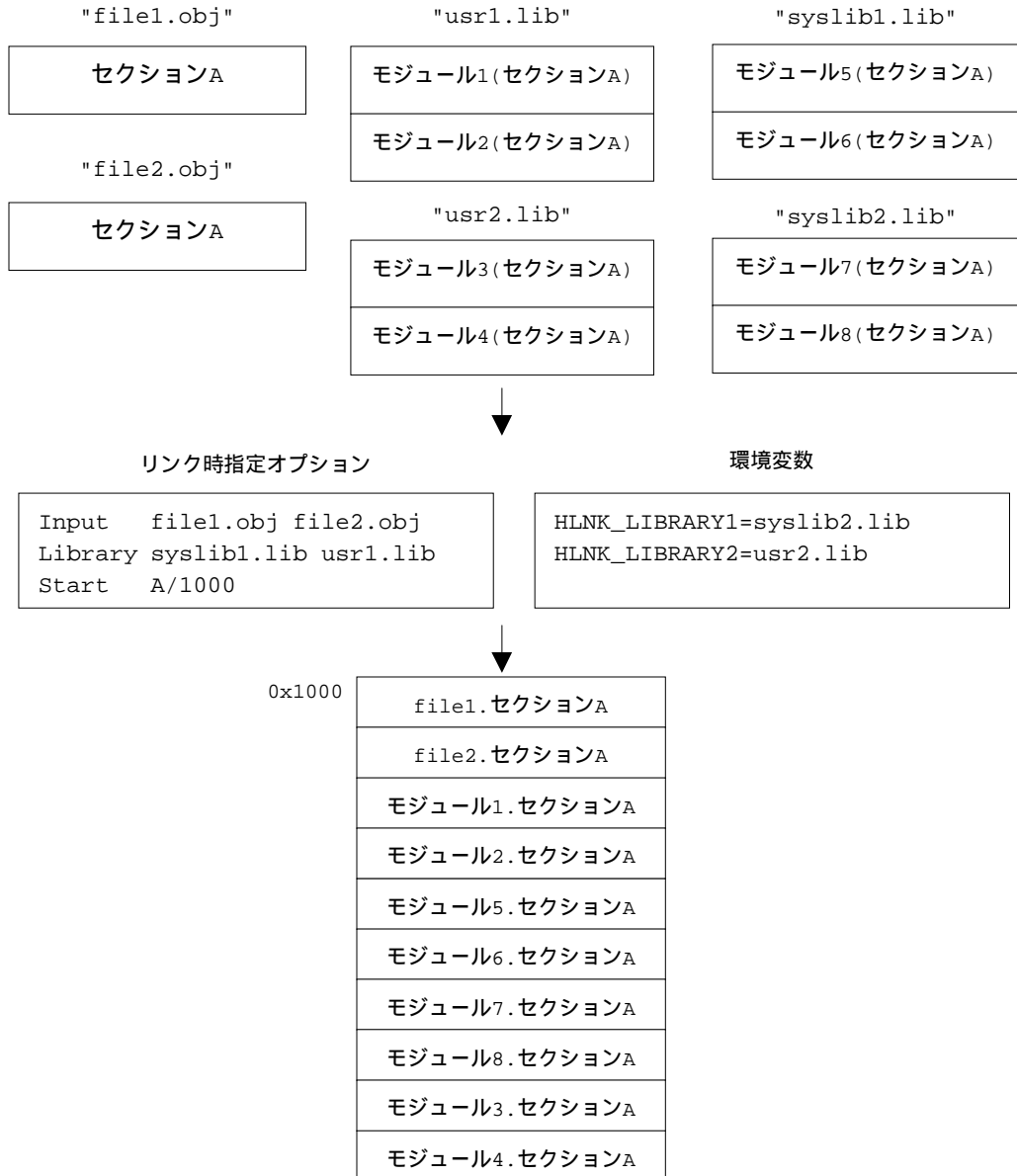


(3) 同名セクションに絶対アドレス形式と相対アドレス形式が含まれている場合、絶対アドレス形式オブジェクトの後に相対アドレス形式オブジェクトを結合します。リロケータブルファイル出力を指定した場合(form=relocate 指定)でも、当該セクションは絶対アドレス形式セクションになります。



(4) 同名セクション内オブジェクトの結合順序に関する規則は以下のとおりです。

- input オプションまたはコマンドライン上の入力ファイル指定順
- library オプションのユーザライブラリ指定順およびライブラリ内モジュール入力順
- library オプションのシステムライブラリ指定順およびライブラリ内モジュール入力順
- 環境変数(HLNK_LIBRARY1 ~ 3)のライブラリ指定順およびライブラリ内モジュール入力順



9.2 初期設定プログラムの作成

本章では、プログラムを SuperH RISC engine マイコンを応用したシステムに組み込む方法を説明します。

プログラムをシステムに組み込むには、以下の準備が必要です。

- メモリの割付け
各セクション、スタック領域、ヒープ領域を、システム上の ROM、RAM のメモリ領域に割り当てる必要があります。
- プログラム実行環境の設定
プログラムの実行環境を設定する処理には、レジスタの初期設定、メモリ領域の初期化、プログラムの起動があります。

また、入出力等の C/C++ライブラリ関数をご使用になる場合は、実行環境の設定時にライブラリの初期化をする必要があります。特に入出力(stdio.h、ios、streambuf、istream、ostream)とメモリ割り付け(stdlib.h、new)の機能を使用する場合は、システムごとに、低水準の入出力ルーチンやメモリ割り付けルーチンを作成する必要があります。

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)を使用する場合も、別途ユーザシステムに合わせてこれらの関数を作成する必要があります

9.2.1 ではプログラムのメモリ領域のアドレスを決定する考え方を説明し、実際にアドレスを決定するための最適化リンケージエディタのオプション指定方法について実例を挙げて説明します。

9.2.2 では実行環境設定の項目を説明し、設定プログラムの実例について説明します。

また、ライブラリ関数の初期設定処理、低水準ルーチンの作成方法および終了処理関数の作成例についても説明します。

9.2.1 メモリ領域の割り付け

本コンパイラの出力したオブジェクトプログラムをシステムに組み込むためには、プログラムの使用するメモリ領域のサイズを決定し、それぞれの領域を適切なメモリアドレスに割り付ける必要があります。

C/C++プログラムが使用するメモリ領域には、C/C++プログラム中の関数に対応する機械語や外部データ定義や静的データメンバで宣言したデータ領域のように静的に割り付ける領域とスタック領域のように動的に割り付ける領域があります。

以下、各領域の割り付け方を説明します。

(1) 静的領域の割り付け

(a) 静的領域の内容

スタック領域、ヒープ領域以外のセクションは静的領域に割り付けます。

C/C++プログラムの各セクション(プログラム領域、定数領域、初期化データ領域、未初期化データ領域、C++初期処理/後処理データ領域、C++仮想関数表領域)は静的領域に割り付けます。

(b) サイズの算出法

静的領域のサイズは、コンパイラ、アセンブラが生成するオブジェクトプログラムサイズとC/C++プログラムが使用するライブラリ関数のサイズの合計になります。

オブジェクトプログラムをリンクしたあと、リンケージリストのリンケージマップ情報にライブラリを含めた各セクションのサイズを出力しますので、静的領域のサイズを知ることができます。

図 9.1 にリンケージリスト内リンケージマップ情報の例を示します。

```
*** Mapping List ***
```

<u>SECTION</u> (1)	<u>START</u> (2)	<u>END</u> (3)	<u>SIZE</u> (4)	<u>ALIGN</u> (5)
P				
C	00000000	000004d6	4d6	2
D	000004d6	00000533	5d	2
B	00000534	0000053c	8	2
	0000053c	00004112	3bd6	2

図 9.1 リンケージリスト内リンケージマップ情報例

コンパイル、アセンブル単位のセクションサイズは、コンパイルリスト内統計情報およびアセンブルリスト内セクション情報に出力されます。図 9.2 にコンパイルリスト内統計情報の例、図 9.3 にアセンブルリスト内セクション情報の例を示します。

```
***** SECTION SIZE INFORMATION *****
```

PROGRAM SECTION(P)	:0x00004A Byte(s)
CONSTANT SECTION(C)	:0x000018 Byte(s)
DATA SECTION(D)	:0x000004 Byte(s)
BSS SECTION(B)	:0x000004 Byte(s)
TOTAL PROGRAM	SECTION: 0000004A Byte(s)
TOTAL CONSTANT	SECTION: 00000018 Byte(s)
TOTAL DATA	SECTION: 00000004 Byte(s)
TOTAL BSS	SECTION: 00000004 Byte(s)
TOTAL PROGRAM SIZE:	0x00006A Byte(s)

図 9.2 コンパイルリスト内統計情報例

*** SECTION DATA LIST			
SECTION	ATTRIBUTE	SIZE	START
P	REL-CODE	000000604	
D	REL-DATA	000000008	
C	REL-DATA	00000005D	
B	REL-DATA	000003BD6	

図 9.3 アセンブルリスト内セクション情報例

標準ライブラリを使用しない場合は、ファイル単位のセクションサイズの合計が静的領域のサイズになります。

標準ライブラリを使用している場合、各セクションのメモリ領域サイズにはライブラリ関数の使用するメモリ領域サイズが加算されます。コンパイラが提供する標準ライブラリの中には、C 言語仕様で規定したライブラリ関数や組み込み関数向け C++ クラスライブラリ以外に、C/C++ プログラムを実行する上で必要な算術演算を行うルーチン(実行時ルーチン)を含みます。そのため、C/C++ ソースプログラム上でライブラリ関数の使用を指定しなくても、標準ライブラリが必要な場合がありますので注意してください。

C/C++ プログラムで使用する実行時ルーチンは、本コンパイラ出力のアセンブリプログラム (code=asmcode 指定) に外部参照シンボルとして出力しますので、そのシンボル名を参照することによって C/C++ プログラムで使用する実行時ルーチン名を知ることができます。以下に具体例を示します。

- C/C++ プログラム例

```
f(int a, int b)
{
    a /= b;
    return a;
}
```

- C コンパイル時に生成されるアセンブリプログラム例

```
.IMPORT  __divls          ;(実行時ルーチンの外部参照宣言)
.EXPORT  _f
.SECTION P,CODE,ALIGN=4

_f:
                                ;function: f
                                ;frame size=4

    STS.L    PR,@-R15
    MOV      R5, R0
    MOV.L    L218,R3            ; __divls
    JSR      @R3
    MOV      R4, R1
    LDS.L    @R15+, PR
    RTS
    NOP

L218:
    .DATA.L  __divls
    .END
```

(c) ROM、RAMの割り付け

プログラムをROM化する場合、セクションの初期値の有無、書き込み操作の可/不可で、ROMに割り付けるかRAMに割り付けるかが決まります。

C/C++プログラムの各セクションをROM化する場合、以下のようにROMとRAMに分けて割り付けます。

- | | | |
|-------------------------------|------------------------------------|----------------|
| • プログラム領域 | (セクション P) | ROM |
| • 定数領域 | (セクション C、\$G0、\$G1 ^{*3}) | ROM |
| • 未初期化データ領域 | (セクション B、\$G0、\$G1 ^{*3}) | RAM |
| • 初期化データ領域 | (セクション D、\$G0、\$G1 ^{*3}) | ROM、RAM((d)参照) |
| • 初期処理/後処理データ領域 ^{*1} | | ROM |
| | (セクションC\$INIT) | |
| • 仮想関数表領域 ^{*2} | (セクション C\$VTBL) | ROM |

【注】*1 C++プログラムでグローバルクラスオブジェクトがあるときにコンパイラが生成します。

*2 C++プログラムで仮想関数宣言があるときにコンパイラが生成します。

*3 \$G0、\$G1 はこれらの領域のいずれか1つのみに配置するようにしてください。

(d) 初期化データ領域の割り付け

初期化データ領域のように、初期値を持ち、プログラム実行時に値の変更が可能なセクションは、リンク時にはROM上に置き、プログラムの実行開始時にRAM上にコピーします。したがって、最適化リンケージエディタのromオプションを用いて、ROM上とRAM上に、二重に領域をとる必要があります。指定例については、「(e)メモリの割り付け例とリンク時のアドレス指定方法」を参照してください。またROM上からRAM上へ値をコピーするセクションの初期設定については、「9.2.2 (2)初期設定」で説明します。

(e) メモリの割り付け例とリンク時のアドレス指定方法

アブソリュートロードモジュール作成時に、最適化リンケージエディタのオプションまたはサブコマンドで各セクション毎に割り付ける領域のアドレスを指定します。以下、静的領域のメモリ割り付け例とリンク時の指定方法について説明します。

図 9.4 に、静的な領域の割り付け例を示します。

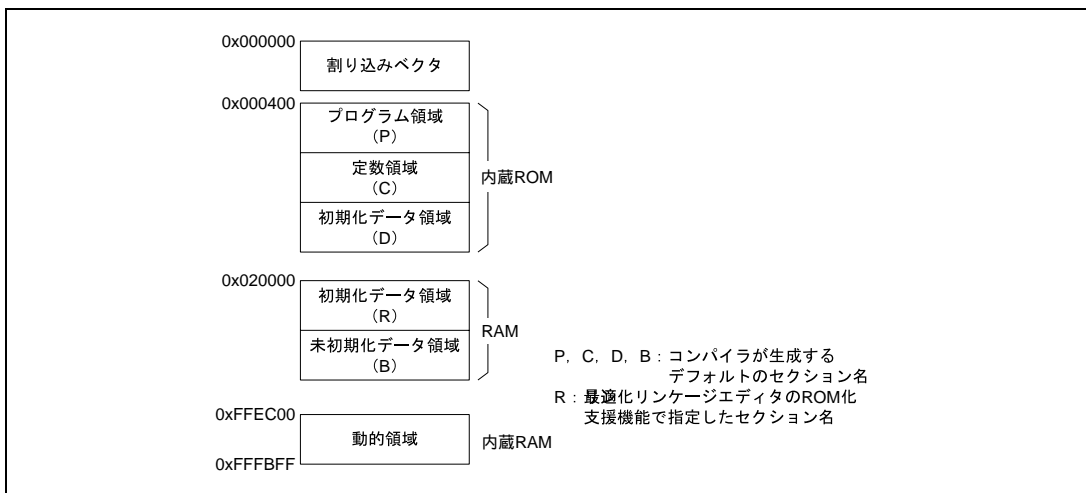


図 9.4 静的な領域の割り付け例

9. プログラミング

図 9.4 に示すメモリ割り付けを行う場合、リンク時に以下のサブコマンドを指定します。

```
ROM D/R .....[1]
START P,C,D/400,R,B/20000 .....[2]
```

- 説明： [1] セクションDと同じ大きさのセクションRを出力ロードモジュールに確保します。また、セクションDに割り付けられたシンボルを参照している場合、セクションR上のアドレスとなるようリロケーションします。セクションDはROM上、セクションRはRAM上の初期化データセクション名になります。
- [2] セクションP、C、Dを内蔵ROMのアドレス0x400から連続した領域に割り付けます。また、セクションR、BをRAMのアドレス0x20000から連続したアドレスに割り付けます。

(2) 動的領域の割り付け

(a) 動的領域の内容

C/C++プログラムで使用する動的領域には、以下の二つがあります。

- スタック領域
- ヒープ領域(メモリ割り付けライブラリ関数で使用)

(b) スタック領域サイズの算出法

C/C++プログラム、標準ライブラリの使用するスタック領域は、最適化リンケージエディタの stack オプションを指定してスタック情報ファイルを出力すると、スタック解析ツールを用いて最大使用量を算出することができます。スタック解析ツールの使用方法については、「6. スタック解析ツール操作方法」を参照してください。

アセンブリプログラムの使用するスタック領域は、スタック解析ツールでは算出できません。以下のC/C++プログラムのスタック使用量算出法を参考にアセンブリプログラムのスタック使用量を算出し、スタック解析ツールで算出したスタック使用量に加算してください。

• C/C++プログラムのスタック使用量計算の考え方

C/C++プログラムの使用するスタック領域は、関数の呼び出しのたびにスタック上に割り付け、関数のリターン時に解放します。スタック領域のサイズを算出するためには、まず各関数ごとのスタック使用量を算出し、関数の呼び出し関係から実際のスタック使用量を算出します。

◆ 各関数の使用するスタック領域

各関数の使用するスタック領域は、コンパイラ出力のオブジェクトリスト中の frame size から分かります。

以下にオブジェクトリストとスタック上の割り付けの具体例を示し、そのスタック使用量の算出法について説明します。

例

次の C プログラムに対するオブジェクトリストとスタック使用量の算出法を示します。C++ プログラムでも同様です。

```
extern int h(char, int *, double );
int h(char a, register int *b, double c)
{
    char *d;

    d= &a;
    h(*d,b,c);
    {
        register int i;

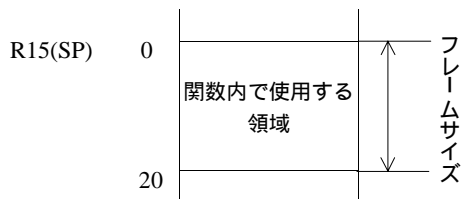
        i= *d;
        return i;
    }
}
```

***** OBJECT LISTING *****

FILE NAME: m0251.c

SCT	OFFSET	CODE	C LABEL	INSTRUCTION	OPERAND	COMMENT
P	00000000		_h:			;function: h
	00000000	2FE6		MOV.L	R14,@-R15	;frame size=20
	00000002	4F22		STS.L	PR,@-R15	
					:	

下位アドレス



上位アドレス

スタック

関数の使用するスタック領域サイズは、フレームサイズの値と同じです。したがって、上記の例で関数 h の使用するスタック領域サイズは、オブジェクトリスト中の項目 COMMENT の frame size の値 20 バイトとなります。

スタック上の引数領域に割り付けられる引数については、「9.3.2(4) 引数とリターン値の設定、参照に関する規則」を参照してください。

9. プログラミング

◆ スタック使用量の算出法

関数呼び出しの関係から使用するスタック領域のサイズを算出します。

例 関数呼び出しの関係と、各関数のスタック使用量の例を図 9.5 に示します。

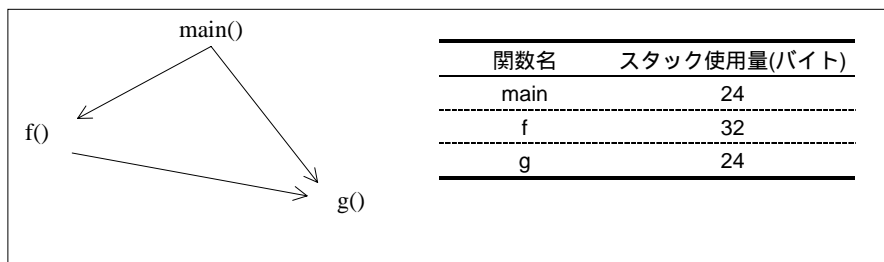


図 9.5 関数呼び出しの関係とスタック使用量の例

この場合、関数「f」を介して関数「g」が呼ばれた時のスタック領域のサイズは、表 9.2 によって計算します。

呼び出し経路	スタックサイズ計(バイト)
main(24) f(32) g(24)	80
main(24) g(24)	48

このように、呼び出しレベルの一番深い関数についてスタック領域のサイズを計算し、その最大値(この場合 80 バイト)のスタック領域を最低限割り付ける必要があります。

【注】 C/C++プログラムの中で再帰呼び出しを行っている場合は、再帰的に呼び出す回数の最大値を算出してから、その関数のスタック領域のサイズに再帰的に呼び出す回数をかけて計算してください。

(c) ヒープ領域

ヒープ領域で使用するメモリ領域のサイズは、C/C++プログラム内でメモリ管理ライブラリ関数 (calloc, malloc, realloc, new 関数)によって割り付ける領域の合計です。ただし、メモリ管理ライブラリ関数は、1回の呼び出しのたびに管理用の領域として4バイト使用します。実際に確保する領域サイズにこの管理領域のサイズを加えて計算してください。

また、コンパイラは、ヒープ領域を1024バイト単位で管理しています。ヒープ領域として確保する領域サイズ(HEAPSIZE)は次のように計算してください。

$$\text{HEAPSIZE} = 1024 \times n \quad (n \geq 1)$$

(メモリ管理ライブラリによって割り付ける領域サイズ) + 管理領域サイズ = HEAPSIZE

入出力ライブラリ関数は、内部処理の中でメモリ管理ライブラリ関数を使用しています。入出力の中で割り付ける領域のサイズは、

516 バイト × (同時にオープンするファイルの数の最大値)
になります。

- 【注】 メモリ管理ライブラリ関数の free、または delete 関数で解放した領域は、再びメモリ管理ライブラリ関数で領域を確保するときに再利用しますが、割り付けを繰り返すことによって空き領域のサイズの合計は十分でも空き領域が小さな領域に分割しているために、新たに要求した大きなサイズの領域を確保できないという状況が生じることがあります。このような状況を避けるために、以下の注意に従ってヒープ領域を使用してください。
- (ア) サイズの大きな領域は、なるべくプログラムの実行開始直後に確保してください。
 - (イ) 解放して再利用するデータ領域のサイズをなるべく一定にしてください。

- 動的領域の割り付け方

動的領域は RAM 上に割り付けます。

スタック領域は、ベクタテーブルにスタック領域の最上位アドレスを SP(スタックポインタ)として設定することにより割り付ける場所が決まります。SH-3、SH-4 では割り込み時の動作が SH-1、SH-2、SH-2E の場合と異なるので、割り込みハンドラが必要になります。

ヒープ領域は、低水準インタフェースルーチン(sbrk)の初期設定で割り付ける場所が決まります。

それぞれ、「9.2.2(1) ベクタテーブルの設定(VEC_TBL)」、「9.2.2(6) 低水準インタフェースルーチン」を参照してください。

9.2.2 実行環境の設定

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。

図 9.6 にプログラムの構成例を示します。

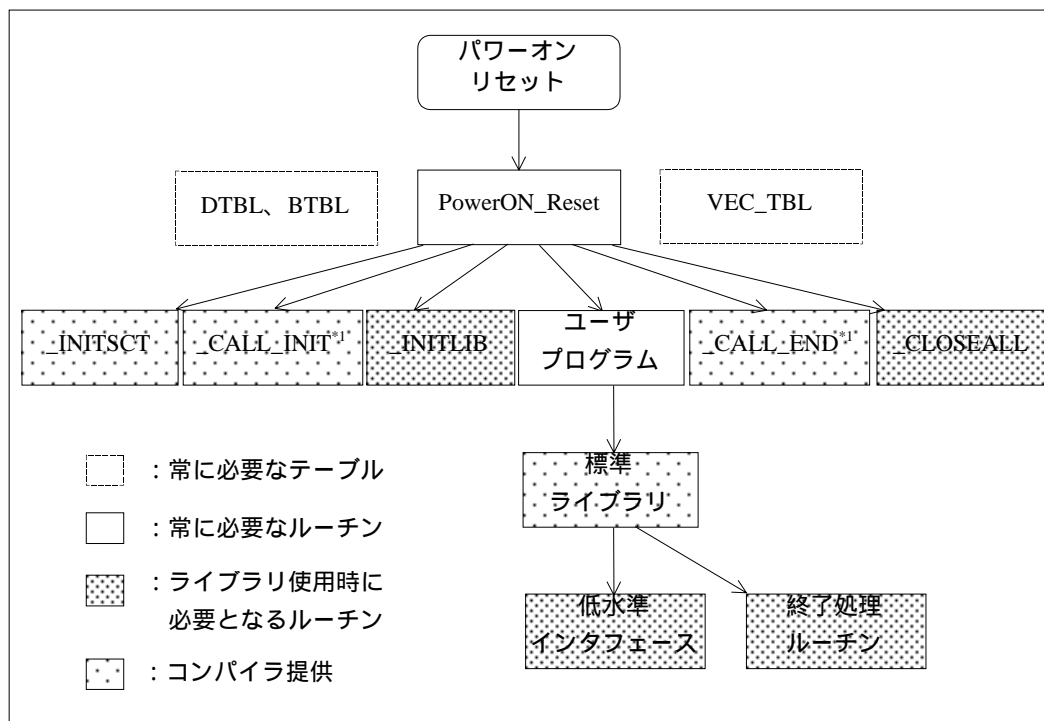


図 9.6 プログラムの構成例

【注】*1 グローバルクラスオブジェクトがあるときに必要になります。

各構成ルーチンの内容は以下のとおりです。

- ベクタテーブルの設定(VEC_TBL)
パワーオンリセットでレジスタの初期設定プログラム(PowerON_Reset)が起動され、またスタックポインタ(SP)に値が設定されるように、ベクタテーブルを設定します。SH-3、SH-4では割り込み時の動作がSH-1、SH-2、SH-2Eの場合と異なるので、割り込みハンドラが必要になります。
- 初期設定(PowerON_Reset)
レジスタの初期設定を行ったあと、初期設定ルーチンを順次呼び出します。
- セクション初期化用テーブル(DTBL、BTBL)
セクションの初期化ルーチンで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。

- セクションの初期化(`_INITSCT`)^{*1}
初期値が設定されていない静的変数領域(未初期化データ領域)をゼロで初期化します。また、初期化データ領域の初期値をROM上からRAM上にコピーします。
- グローバルクラスオブジェクト初期処理(`_CALL_INIT`)^{*1*2}
グローバルに宣言されたクラスオブジェクトに対してコンストラクタを呼び出します。
- グローバルクラスオブジェクト後処理(`_CALL_END`)^{*1*2}
main 関数の実行後、グローバルクラスオブジェクトに対してデストラクタを呼び出します。
- C/C++ライブラリの初期設定(`_INITLIB`)
ライブラリ関数の中で、初期設定の必要なものについて、初期設定を行います。特に、標準入出力を行う準備をします。
- ファイルのクローズ(`_CLOSEALL`)
オープンしているファイルをすべてクローズします。
- 低水準インタフェースルーチン
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`)^{*3}
プログラムの終了処理を行います。

【注】 *1: 標準ライブラリとして提供しています。

*2: C++プログラム中にグローバルクラスオブジェクトの宣言がある時に必要な処理です。

*3: 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。

以下、この構成に従って各処理の実現方法について解説します。

(1) ベクタテーブルの設定(`VEC_TBL`)

パワーオンリセットで、レジスタの初期設定を行う関数「`PowerON_Reset`」が呼び出されるようにするためには、ベクタテーブルの0番地に関数「`PowerON_Reset`」の先頭アドレスを設定します。また、スタックポインタ(SP)を設定するためには4番地にスタック領域の最上位アドレスを設定します。SH-3、SH-4では割り込み時の動作がSH-1、SH-2、SH-2Eの場合と異なるので割り込みハンドラが必要になります。

また、ユーザシステムで割り込み処理を使用する場合は、割り込みベクタの設定も本ルーチンで行います。以下にそのコーディング例を示します。


```
.GLOBAL  INT_TLBMiss_Store
;H'080 Initial page write
.GLOBAL  INT_TLBInitial_Page
;H'0A0 TLB protect (load)
.GLOBAL  INT_TLBProtect_Load
;H'0C0 TLB protect (store)
.GLOBAL  INT_TLBProtect_Store
;H'0E0 Address error (load)
.GLOBAL  INT_Address_load
;H'100 Address error (store)
.GLOBAL  INT_Address_store
;H'120 Reserved
.GLOBAL  INT_Reserved1
;H'140 Reserved
.GLOBAL  INT_Reserved2
;H'160 TRAPA
.GLOBAL  INT_TRAPA
;H'180 Illegal code
.GLOBAL  INT_Illegal_code
;H'1A0 Illegal slot
.GLOBAL  INT_Illegal_slot
;H'1C0 NMI
.GLOBAL  INT_NMI
;H'1E0 User breakpoint trap
.GLOBAL  INT_User_Break
;H'200 External hardware interrupt
.GLOBAL  INT_Extern_0000
;H'220 External hardware interrupt
.GLOBAL  INT_Extern_0001
;H'240 External hardware interrupt
.GLOBAL  _INT_Extern_0010
;H'260 External hardware interrupt
.GLOBAL  _INT_Extern_0011
;H'280 External hardware interrupt
.GLOBAL  _INT_Extern_0100
;H'2A0 External hardware interrupt
.GLOBAL  _INT_Extern_0101
;H'2C0 External hardware interrupt
.GLOBAL  _INT_Extern_0110
;H'2E0 External hardware interrupt
.GLOBAL  _INT_Extern_0111
;H'300 External hardware interrupt
.GLOBAL  _INT_Extern_1000
;H'320 External hardware interrupt
.GLOBAL  _INT_Extern_1001
;H'340 External hardware interrupt
.GLOBAL  _INT_Extern_1010
;H'360 External hardware interrupt
.GLOBAL  _INT_Extern_1011
;H'380 External hardware interrupt
.GLOBAL  _INT_Extern_1100
;H'3A0 External hardware interrupt
.GLOBAL  _INT_Extern_1101
;H'3C0 External hardware interrupt
.GLOBAL  _INT_Extern_1110
;H'3E0 External hardware interrupt
.GLOBAL  _INT_Extern_1111
```

9. プログラミング

```
;H'400 TMU0 TUNIO
.GLOBAL    _INT_Timer_Under_0
;H'420 TMU1 TUNI1
.GLOBAL    _INT_Timer_Under_1
;H'440 TMU2 TUNI2
.GLOBAL    _INT_Timer_Under_2
;H'460 TMU2 TICPI2
.GLOBAL    _INT_Input_Capture
;H'480 RTC ATI
.GLOBAL    _INT_RTC_ATI
;H'4A0 RTC PRI
.GLOBAL    _INT_RTC_PRI
;H'4C0 RTC CUI
.GLOBAL    _INT_RTC_CUI
;H'4E0 SCI ERI
.GLOBAL    _INT_SCI_ERI
;H'500 SCI RXI
.GLOBAL    _INT_SCI_RXI
;H'520 SCI TXI
.GLOBAL    _INT_SCI_TXI
;H'540 SCI TEI
.GLOBAL    _INT_SCI_TEI
;H'560 WDT ITI
.GLOBAL    _INT_WDT
;H'580 REF RCMI
.GLOBAL    _INT_REF_RCMI
;H'5A0 REF ROVI
.GLOBAL    _INT_REF_ROVI

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;                               vhandler.src                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.INCLUDE   "env.inc"
.INCLUDE   "vect.inc"

IMASKclr:
.EQU      H'FFFFFF0F
RBBLclr:
.EQU      H'CFFFFFFF
MDRBBLset:
.EQU      H'70000000

.IMPORT    RESET_Vectors
.IMPORT    INT_Vectors
.IMPORT    INT_MASK

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;                               macro definition                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.MACRO     PUSH_EXP_BASE_REG
STC.L     SSR,@-R15      ; save SSR
STC.L     SPC,@-R15     ; save SPC
STS.L     PR,@-R15      ; save CONTEXT REGISTERS
STC.L     R7_BANK,@-R15
STC.L     R6_BANK,@-R15
STC.L     R5_BANK,@-R15
```

```

STC.L    R4_BANK,@-R15
STC.L    R3_BANK,@-R15
STC.L    R2_BANK,@-R15
STC.L    R1_BANK,@-R15
STC.L    R0_BANK,@-R15
.ENDM

;

.MACRO POP_EXP_BASE_REG
LDC.L    @R15+,R0_BANK    ; RECOVER REGISTERS
LDC.L    @R15+,R1_BANK
LDC.L    @R15+,R2_BANK
LDC.L    @R15+,R3_BANK
LDC.L    @R15+,R4_BANK
LDC.L    @R15+,R5_BANK
LDC.L    @R15+,R6_BANK
LDC.L    @R15+,R7_BANK
LDS.L    @R15+,PR
LDC.L    @R15+,SPC
LDC.L    @R15+,SSR
.ENDM

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                reset                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.SECTION  RSTHandler, CODE
_ResetHandler:
MOV.L    #EXPEVT,R0
MOV.L    @R0,R0
SHLR2    R0
SHLR     R0
MOV.L    #_RESET_Vectors,r1
ADD      R1,R0
MOV.L    @R0,R0
JMP      @R0
NOP

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                exceptional interrupt                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.SECTION  INTHandler, CODE
.EXPORT   INTHandlerPRG
INTHandlerPRG:
_ExpHandler:
PUSH_EXP_BASE_REG

;

MOV.L    #EXPEVT,R0    ; set event address
MOV.L    @R0,R1        ; set exception code
MOV.L    #_INT_Vectors,R0 ; set vector table address
ADD      #-(H'40),R1   ; exception code - H'40
SHLR2    R1
SHLR     R1
MOV.L    @(R0,R1),R3   ; set interrupt function addr

;

MOV.L    #_INT_MASK,R0 ; interrupt mask table addr
SHLR2    R1
MOV.B    @(R0,R1),R1   ; interrupt mask
EXTU.B   R1,R1

```

9. プログラミング

```

;
    STC        SR,R0          ; save SR
    MOV.L     #(RBBLclr&IMASKclr),R2
                                ; RB,BL,mask clear data
    AND       R2,R0          ; clear mask data
    OR        R1,R0          ; set interrupt mask
    LDC       R0,SSR         ; set current status
;
    LDC.L     R3,SPC
    MOV.L     #__int_term,R0 ; set interrupt terminate
    LDS       R0,PR
;
    RTE
    NOP
;
    .POOL
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                               Interrupt terminate                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    .ALIGN    4
__int_term:
    MOV.L     #MDRBBLset,R0   ; set MD,BL,RB
    LDC.L     R0,SR
;
    POP_EXP_BASE_REG
;
    RTE                               ; return
    NOP
;
    .POOL
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                               TLB miss interrupt                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    .ORG      H'300
__TLBmissHandler:
    PUSH_EXP_BASE_REG
;
    MOV.L     #EXPEVT,R0      ; set event address
    MOV.L     @R0,R1          ; set exception code
    MOV.L     #_INT_Vectors,R0 ; set vector table address
    ADD       #-(H'40),R1     ; exception code - H'40
    SHLR2    R1
    SHLR     R1
    MOV.L     @(R0,R1),R3     ; set interrupt function addr
;
    MOV.L     #_INT_MASK,R0   ; interrupt mask table addr
    SHLR2    R1
    MOV.B     @(R0,R1),R1     ; interrupt mask
    EXTU.B   R1,R1
;
    STC        SR,R0          ; save SR
    MOV.L     #(RBBLclr&IMASKclr),R2
                                ; RB,BL,mask clear data

```

```

        AND        R2,R0            ; clear mask data
        OR         R1,R0            ; set interrupt mask
        LDC        R0,SSR           ; set current status
;
        LDC.L      R3,SPC
        MOV.L      #__int_term,R0   ; set interrupt terminate
        LDS        R0,PR
;
        RTE
        NOP
;
        .POOL
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                     IRQ                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        .ORG        H'500
_IRQHandler:
        PUSH_EXP_BASE_REG
;
        MOV.L      #INTEVT,R0       ; set event address
        MOV.L      @R0,R1           ; set exception code
        MOV.L      #_INT_Vectors,R0 ; set vector table address
        ADD        #-(H'40),R1      ; exception code - H'40
        SHLR2      R1
        SHLR       R1
        MOV.L      @(R0,R1),R3      ; set interrupt function addr
;
        MOV.L      #_INT_MASK,R0    ; interrupt mask table addr
        SHLR2      R1
        MOV.B      @(R0,R1),R1      ; interrupt mask
        EXTU.B     R1,R1
;
        STC        SR,R0            ; save SR
        MOV.L      #(RBBLclr&IMASKclr),R2
;                                     ; RB,BL,mask clear data
        AND        R2,R0            ; clear mask data
        OR         R1,R0            ; set interrupt mask
        LDC        R0,SSR           ; set current status
;
        LDC.L      R3,SPC
        MOV.L      #__int_term,R0   ; set interrupt terminate
        LDS        R0,PR
;
        RTE
        NOP
;
        .POOL
        .END

```

【注】 #pragma interrupt を指定した関数をリンクしないでください。

(2) 初期設定(PowerON_Reset)

ライブラリ関数を使用する場合には、本関数でライブラリの初期設定を行う「_INITLIB」とファイルのクローズ処理を行う「_CLOSEALL」を呼び出します。以下に「PowerON_Reset」のコーディング例を示します。SH-3、SH-4 では割り込み時の動作が SH-1、SH-2、SH-2E の場合と異なるので割り込みハンドラが必要になります。

例

```
#include <_h_c_lib.h>
#include <machine.h>

#pragma entry PowerON_Reset_PC
#pragma stacksize 0x100

#define SR_Init      0x000000F0
#define INT_OFFSET  0x10

extern unsigned int INT_Vectors;
extern void PowerON_Reset_PC();
extern void main();
#ifdef __cplusplus
extern "C" {
#endif
extern void _INIT_IOLIB();
extern void _INIT_OTHERLIB();
extern void _CLOSEALL();
#ifdef __cplusplus
}
#endif

void PowerON_Reset_PC() {
    set_vbr((void *)(INT_Vectors - INT_OFFSET));

    _INITSCT();
    _INIT_IOLIB();
    _INIT_OTHERLIB();
#ifdef __cplusplus
    _CALL_INIT();
#endif
    set_cr(SR_Init);
    main();
#ifdef __cplusplus
    _CALL_END();
#endif
    _CLOSEALL();
    sleep();
}
```


(3) セクション初期化用テーブル(DTBL、BTBL)

セクションの初期化ルーチン(_INITISCT)では、未初期化データセクションをゼロで初期化し、初期化データセクションの ROM 上にある初期化データを RAM 上にコピーします。ここでは、_INITISCT 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、セクションの初期化用テーブルに設定します。

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 $DSEC          //セクション名を C$DSEC にします
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 $BSEC          //セクション名を C$BSEC にします
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
} DTBL[] = { _sectop("B"), _secend("B")};
```

(4) C/C++ライブラリ関数の初期設定(_INITLIB)

ここでは、C/C++ライブラリ関数の初期設定方法を説明します。

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。

- <stdio.h>、<ios>、<streambuf>、<istream>、<ostream>の各関数と assert マクロを使用する場合、標準入出力の初期設定(_INIT_IOLIB)が必要です。
- 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。また、図 9.7 に FILE 型データを示します。

9. プログラミング

```
#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:1032)
const int _nfiles = IOSTREAM;           // 入出力ファイル数を指定します(省略時:20)
struct _iobuf _iob[IOSTREAM];
unsigned char sml_buf[IOSTREAM];
extern char *_slpstr;

#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)
{
    // 低水準ライブラリに必要な初期設定をしてください
}

void _INIT_IOLIB(void)
{
    FILE *fp;
    for( fp = _iob; fp < _iob + _nfiles; fp++ ) // FILE 型データの初期設定です
    {
        fp->_bufptr = NULL;
        fp->_bufcnt = 0;
        fp->_buflen = 0;
        fp->_bufbase = NULL;
        fp->_ioflag1 = 0;
        fp->_ioflag2 = 0;
        fp->_iofd = 0;
    }
    if(freopen("stdin" *1, "r", stdin)== NULL) // 標準入力ファイルをオープンします
        stdin->_ioflag1 = 0xff; // オープン失敗時ファイルアクセスを禁止します
    stdin->_ioflag1 |= _IOUNBUF; // データのバッファリングなしに設定します*2
    if(freopen("stdout" *1, "w", stdout)== NULL) // 標準出力ファイルをオープンします
        stdout->_ioflag1 = 0xff; // オープン失敗時ファイルアクセスを禁止します
    stdout->_ioflag1 |= _IOUNBUF; // データのバッファリングなしに設定します*2
    if(freopen("stderr" *1, "w", stderr)== NULL) // 標準エラーファイルをオープンします
        stderr->_ioflag1 = 0xff; // オープン失敗時ファイルアクセスを禁止します
    stderr->_ioflag1 |= _IOUNBUF; // データのバッファリングなしに設定します*2
}

void _INIT_OTHERLIB(void)
{
    srand(1); // rand 関数を使用する場合の初期設定です
    _slpstr=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。
- *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。

```

/* ファイル型データのC言語での宣言 */

struct _iobuf{
    unsigned char * _bufptr; /* バッファへのポインタ */
    long _bufcnt; /* バッファカウント */
    unsigned char * _bufbase; /* バッファへのベースポインタ */
    long _buflen; /* バッファ長 */
    char _ioflag1; /* I/Oフラグ */
    char _ioflag2; /* I/Oフラグ */
    char _iofd; /* I/Oフラグ */
}iob[_nfiles];

```

図 9.7 FILE 型データ

(5) ファイルのクローズ(_CLOSEALL)

通常ファイルへの出力は、メモリ領域上のバッファにためておき、バッファが一杯になったときに実際の外部記憶装置への書き出しを行います。したがってファイルのクローズを行わないと、ファイルへの出力内容が外部記憶装置へ書き出されないことがあります。

機器組み込み用のプログラムの場合、通常プログラムが終了することはありません。しかし、プログラムの誤りなどにより main 関数が終了する場合、オープンしているファイルは、すべてクローズしなければなりません。

本処理は、main 関数終了時にオープンしているファイルのクローズを行います。

ファイルのクローズを行うプログラム例を以下に示します。

```

#include <stdio.h>

#ifdef __cplusplus
extern "C"
#endif
void _CLOSEALL(void)
{
    int i;

    for( i=0; i < _nfiles; i++ )

        // ファイルがオープンしているかどうかチェックします
        if( _iob[i]._ioflag1 & (_IOREAD | _IOWRITE | _IORW ) )
            fclose( &_iob[i] ); // ファイルをクローズします
}

```

9. プログラミング

(6) 低水準インタフェースルーチン

標準入出力、メモリ管理ライブラリを C/C++ プログラムで使用する場合は、低水準インタフェースルーチンを作成しなければなりません。表 9.3 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 9.3 低水準インタフェースルーチンの一覧

	名称	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	error_addr	errno アドレスの取得
8	wait_sem	セマフォの確保
9	signal_sem	セマフォの解放

【注】 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に必要な初期化は、プログラム起動時に行う必要があります。これは、「9.2.2(4) C/C++ライブラリ関数の初期設定(_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。

【注】 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語です。ユーザプログラム中では使用しないでください。

(a) 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をする場合はバッファの位置、サイズ等の情報。
- ディスクファイルならば、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。

(b) 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原型宣言してください。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：

簡易説明

(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の意味を示します)
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の意味を示します)

int open(char *name, int mode, int flg)

説明 第1引数のファイル名に対応するファイル进行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。

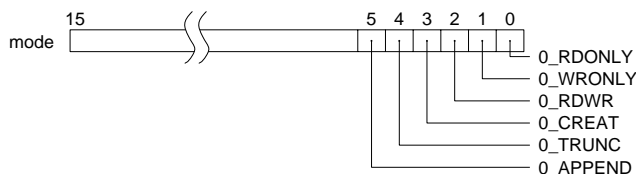


表 9.4 open ルーチン mode ビット説明

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)

ファイルクローズ

int close(int 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

データ読み込み

int read(int fileno, char *buf, unsigned int 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

データ書き出し

int write(int fileno, char *buf, unsigned int count)

説明	第 2 引数 (<i>buf</i>) の指す領域から、第 1 引数 (<i>fileno</i>) 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 (<i>count</i>) 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、 <i>count</i> 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	<i>fileno</i>	書き出しの対象となるファイル番号
	<i>buf</i>	書き出すデータ領域
	<i>count</i>	書き出すバイト数

ファイル内位置設定

int lseek(int fileno, long offset, int 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 (<i>base</i>) によって、以下の方法で計算し設定してください。	
	(1) <i>base</i> が 0 のとき	ファイルの先頭から <i>offset</i> バイトの位置に設定します。
	(2) <i>base</i> が 1 のとき	現在の位置に <i>offset</i> バイトを加えた位置に設定します。
	(3) <i>base</i> が 2 のとき	ファイルのサイズに <i>offset</i> 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こ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	<i>fileno</i>	対象となるファイル番号
	<i>offset</i>	読み込み / 書き出しの位置を示すオフセット (バイト単位)
	<i>base</i>	オフセットの起点

メモリ領域割り付け

char *sbrk(int 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 <code>sbrk</code> 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、「 <code>(char *)-1</code> 」を返してください。	
リターン値	正常：	割り付けた領域の先頭アドレス
	異常：	<code>(char *)-1</code>
引数	<code>size</code>	割り付けるデータのサイズ

errno アドレス取得***int *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 <code>reent</code> 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	現在のタスクが持つエラー番号のアドレス

int wait_sem (int 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 <code>reent</code> 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

int signal_sem (int 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 <code>reent</code> 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

(c) 低水準インタフェースルーチンコーディング例

```

/*****
/*                               lowsrc.c:                               */
/*-----*/
/*   SuperH RISC engine シリーズ 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        /* 書き込み専用 */
#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 HEAPSIZ 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 */

static union {
    long dummy;                /* 4バイト境界にするためのダミー */
    char heap[HEAPSIZ];        /* sbrk で管理する領域の宣言 */
} heap_area;

static char *brk=(char*)&heap_area; /* sbrk で割り付けた領域の最終アドレス */

```

9. プログラミング

```

/*****
/*
/*          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)                */
*****/
int open(char *name,          /* ファイル名          */
         int mode)          /* ファイルのモード */
{
    /* 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)                */
*****/
int close(int 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： 実際読み込んだ文字数 (成功)          */
/*          -1          (失敗)          */
/*****
int read(int fileno,          /* ファイル番号          */
        char *buf,          /* 転送先バッファアドレス */
        unsigned int count) /* 読み込み文字数          */
{
    unsigned int 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： 実際書き出した文字数 (成功)          */
/*          -1          (失敗)          */
/*****
int write(int fileno,          /* ファイル番号          */
         char *buf,          /* 転送元バッファアドレス */
         unsigned int count) /* 書き出し文字数          */
{
    unsigned int i;
    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;
    }
}

```

9. プログラミング

```
/******  
/*          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(int fileno,          /* ファイル番号          */  
           long offset,          /* 読み込み / 書き出し位置          */  
           int 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(unsigned long 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          ;  
; SuperH RISC engine シリーズ シミュレータ・デバッガ インタフェースルーチン          ;  
;          - 一文字入出力を行います -          ;  
;*****  
    .EXPORT    _charput  
    .EXPORT    _charget  
SIM_IO:  
    .EQU      H'0000          ;TRAP_ADDRESS の指定  
  
    .SECTION  P, CODE, ALIGN=4
```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;           _charput: 一文字出力
;           C プログラムインタフェース: charput(char)
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_charput:
    MOV.L    O_PAR,R0        ;バッファアドレスの設定
    MOV.B    R4,@R0         ;パラメタをバッファに設定
    MOV.L    #O_PAR,R1      ;パラメタブロックアドレスの設定
    MOV.L    #H'01220000,R0 ;機能コードの設定(putc)
    MOV.W    #SIM_IO,R2     ;システムコールアドレスの設定
    JSR      @R2
    NOP
    RTS
    NOP

    .ALIGN   4
O_PAR:                                ;パラメタブロック領域
    .DATA.L  OUT_BUF

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;           _charget: 一文字入力
;           C プログラムインタフェース: char charget(void)
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    .ALIGN   4
_charget:
    MOV.L    #I_PAR,R1      ;パラメタブロックアドレスの設定
    MOV.L    #H'01210000,R0 ;機能コードの設定(getc)
    MOV.W    #SIM_IO,R2     ;システムコールアドレスの設定
    JSR      @R2
    NOP
    MOV.L    I_PAR,R0       ;バッファアドレスの設定
    MOV.B    @R0,R0        ;入力データを返却値に設定
    RTS
    NOP

    .ALIGN   4
I_PAR:                                ;パラメタブロック領域
    .DATA.L  IN_BUF

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;           入出力バッファの定義
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    .SECTION  B,DATA,ALIGN=4

OUT_BUF:
    .RES.L   1              ;出力バッファ
IN_BUF:
    .RES.L   1              ;入力バッファ

    .END

```

9. プログラミング

(d) 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。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を使用する場合には必要になります。

`wait_sem` 関数、`signal_sem` 関数で `NG` が返った場合、`errno` に以下を設定し、ライブラリ関数からリターンします。

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
	EIOBRESM	_iob 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
	EIBFRSM	_iob 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      /* malloc 用セマフォ No.      */
#define STRTOK_SEM     2      /* strtok 用セマフォ No.     */
#define FILE_TBL_SEM  3      /* _iob 用セマフォ No.       */
#define SEMSIZE        4
#define TRUE           1
#define FALSE          0
#define OK              1
#define NG              0

extern int *errno_addr(void);
extern int wait_sem(int);
extern int signal_sem(int);

int sem_errno;
int force_fail_signal_sem = FALSE;
static int semaphore[SEMSIZE];

/*****
/*          errno_addr:errno アドレスの取得
/*          リターン値：errno アドレス
*****/
int *errno_addr(void)
{
    /* 現在のタスクの errno アドレスを返してください */
    return (&sem_errno);
}

/*****
/*          wait_sem:指定されたセマフォの確保
/*          リターン値：OK(=1) (成功)
/*          NG(=0) (失敗)
*****/
int wait_sem(int semnum)      /* セマフォ ID      */
{
    if((0 <= semnum) && (semnum < SEMSIZE)) {
        if(semaphore[semnum] == FALSE) {
            semaphore[semnum] = TRUE;
            return(OK);
        }
    }
    return(NG);
}
```



```

/*****
/*          signal_sem:指定されたセマフォの解放          */
/*          リターン値：OK(=1) (成功)                    */
/*          NG(=0) (失敗)                                */
/*****
int signal_sem(int semnum) /* セマフォ ID */
{
    if(!force_fail_signal_sem) {
        if((0 <= semnum) && (semnum < SEMSIZE)) {
            if( semaphore[semnum] == TRUE ) {
                semaphore[semnum] = FALSE;
                return(OK);
            }
        }
    }
    return(NG);
}

```

(7) 終了処理ルーチン

(a) 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 NULL を返します。そうでなければ NULL 以外の値(ここでは、登録した関数のアドレス)を返します。

以下にプログラム例を示します。

例:

```
#include <stdlib.h>
typedef void *atexit_t ;

int _atexit_count=0 ;

atexit_t (*_atexit_buf[32])(void) ;

#ifdef __cplusplus
extern "C"
#endif
atexit_t atexit(atexit_t (*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 NULL ;
    if (_atexit_count==32)                   // 登録数の限界値をチェックします
        return NULL ;
    else {
        _atexit_buf[_atexit_count++]=f;     // 関数のアドレスを登録します
        return f;
    }
}
```

(b) 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」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>

typedef void * atexit_t ;
extern int _atexit_count ;
extern atexit_t (*_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
}

```

(c) 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

9. プログラミング

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
}
```

9.3 C/C++プログラムとアセンブリプログラムとの結合

C/C++プログラムとアセンブリプログラムの結合時に留意すべき以下の内容について述べます。

- 外部名の相互参照方法
- 関数呼び出しのインタフェース

9.3.1 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。

- グローバル変数であって、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)

(1) アセンブリプログラムの外部名を C/C++プログラムで参照する方法

アセンブリプログラムでは、.EXPORT 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。

アセンブリプログラム(定義する側)	C/C++プログラム(参照する側)
.EXPORT _a, _b	extern int a,b;
.SECTION D,DATA,ALIGN=4	
_a: .DATA.L 1	void f()
_b: .DATA.L 1	{
.END	a+=b;
	}

(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から参照する方法

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、.IMPORT 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。

C/C++プログラム(定義する側)	アセンブリプログラム(参照する側)
int a;	.IMPORT _a
	.SECTION P,CODE,ALIGN=2
	MOV.L A_a,R1
	MOV.L @R1,R0
	ADD #1,R0
	RTS
	MOV.L R0,@R1
	.ALIGN 4
	A_a: .DATA.L _a
	.END

9. プログラミング

- (3) C++プログラムの外部(関数)名をアセンブリプログラムから参照する方法
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。

C++プログラム(呼び出される側)

```
extern "C"  
void sub ( )  
{  
    :  
}
```

アセンブリプログラム(呼び出す側)

```
.IMPORT _sub  
.SECTION P, CODE, ALIGN=4  
:  
  
STS.L PR, @-R15  
MOV.L R1, @(1, R15)  
MOV R3, R12  
MOV.L A_sub, R0  
JSR @R0  
NOP  
LDS.L @R15+, PR  
:  
A_sub: .DATA.L _sub  
.END
```

9.3.2 関数呼び出しのインタフェース

C/C++プログラムとアセンブリプログラム間で相互に関数呼び出しを行うときに、アセンブリプログラム側で守るべき次の4つの規則について説明します。

- (1) スタックポインタに関する規則
- (2) スタックフレームの割り付け、解放に関する規則
- (3) レジスタに関する規則
- (4) 引数とリターン値の設定、参照に関する規則

(1) スタックポインタに関する規則

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、解放に関する規則

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。

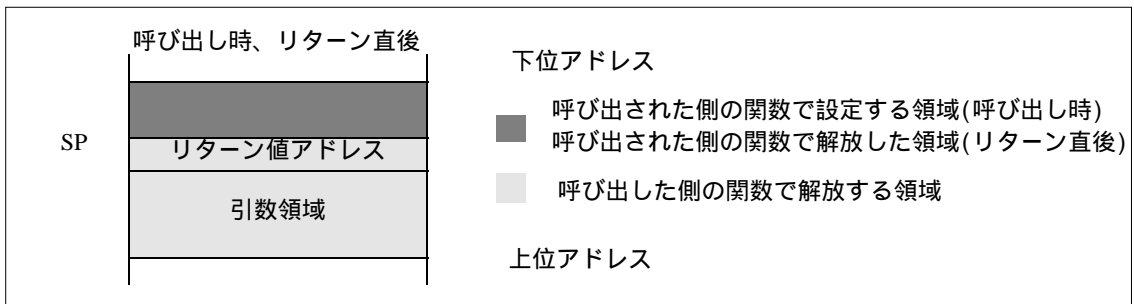


図 9.8 スタックフレームの割り付け、解放に関する規則

(3) 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レジスタと保証しないレジスタがあります。レジスタの保証規則を表 9.5 に示します。

表 9.5 関数呼び出し前後のレジスタ保証規則

項目	対象レジスタ	プログラミングにおける注意点
1 保証しないレジスタ	R0 ~ R7 FR0 ~ FR11 ^{*1} DR0 ~ DR10 ^{*2} FPUL ^{*1,*2} , FPSCR ^{*1,*2}	関数呼び出し時に対象レジスタに有効な値があれば、呼び出し側で値を退避する。呼び出される側の関数では退避せずに使用できる。ただし、fpscr=safe を指定した場合は FPSCR は保証するレジスタ。
2 保証するレジスタ	R8 ~ R15 MACH, MACL, PR FR12 ~ FR15 ^{*1} DR12 ~ DR14 ^{*2}	対象レジスタのうち関数内で使用するレジスタの値を退避し、リターン時に回復する。ただし、macsave = 0 を指定した場合は MACH, MACL は保証しないレジスタ。また、gbr=auto を指定した場合は GBR は保証するレジスタ。

【注】 *1 SH-2E、SH-4 の単精度浮動小数点用レジスタです。

*2 SH-4 の倍精度浮動小数点用レジスタです。

9. プログラミング

以下、レジスタの保証規則の具体例を示します。

- (a) アセンブリプログラムのサブルーチンを C/C++プログラムから呼び出す場合
アセンブリプログラム(呼び出される側)

```
.EXPORT _sub
.SECTION P, CODE, ALIGN=4
_sub: MOV.L   R14, @-R15           ; 関数内で使用するレジスタの退避
      MOV.L   R13, @-R15
      ADD     #-8, R15

      :                           ; 関数本体の処理
                                       ; (R0 ~ R7は保証しないレジスタのため、
                                       ; 関数内では退避せずに使用可能)

      ADD     #8, R15
      MOV.L   @R15+, R13         ; 退避したレジスタの回復
      RTS
      MOV.L   @R15+, R14
.END
```

C/C++プログラム(呼び出す側)

```
#ifdef __cplusplus
extern "C"
#endif
void sub();

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


(b) C/C++プログラムの関数をアセンブリプログラムから呼び出す場合

C/C++プログラム(呼び出される側)

```
void sub()
{
    :
}
```

アセンブリプログラム(呼び出す側)

```
.IMPORT _sub
.SECTION P, CODE, ALIGN=4
:

STS.L PR, @-R15
MOV.L R1, @(1, R15)
MOV R3, R12
MOV.L A_sub, R0
JSR @R0
NOP
LDS.L @R15+, PR
:
A_sub: .DATA.L _sub
.END
```

呼び出す関数名の先頭に下線"_"を付けたものを.IMPORTで宣言(C)
コンパイラが関数宣言/定義から生成した外部名を.IMPORTで宣言(C++)

関数呼び出しする場合は、PRレジスタ(リターンアドレス格納レジスタ)を退避
レジスタR0~R7に有効な値があれば空きレジスタ(R8~R14)またはスタックに退避
関数「sub」の呼び出し

PRレジスタの復帰

関数「sub」のアドレスデータ

【注】 * 関数名、静的データメンバから生成する外部名は、C++コンパイルのとき一定の規則で変換を行っています。コンパイラが生成した外部名を知る必要があるときは、code=asmcode または listfile オプションにてコンパイラが生成する外部名を参照してください。また、C++の関数を「extern "C"」を付与して関数定義を行えば、外部名はCの関数と同様の生成規則になります。ただし、その関数を多重定義できなくなります。

(4) 引数とリターン値の設定、参照に関する規則

以下、引数とリターン値の設定、参照方法について説明します。解説では、まず引数とリターン値に対する一般的な規則について述べたあと、引数の割り付け方とリターン値の設定場所について述べます。

(a) 引数とリターン値に対する一般的な規則

• 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。

- 型変換の規則

引数を渡す場合、またはリターン値を返す場合、自動的に型変換を行う場合があります。以下、この型変換の規則について説明します。

- ◆ 型の宣言された引数の型変換

原型宣言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。

- ◆ 型の宣言されていない引数の型変換

原型宣言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。

- (signed) char 型、unsigned char 型、(signed) short 型、unsigned short 型の引数は、(signed) int 型に変換します。
- 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。

- ◆ 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

(1)

```
long f( );
long f( )
{
    float x;
    return x;    ← 原型宣言にしたがってリターン値はlong型に変換されます。
}
```

(2)

```
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、int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、int型に変換されます。

(b) 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レジスタに割り付けられないとき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 9.9 に、引数割り付け領域の一般規則を表 9.6 にそれぞれ示します。C++プログラムの非静的関数メンバの this ポインタは、R4 に割り付けられます。

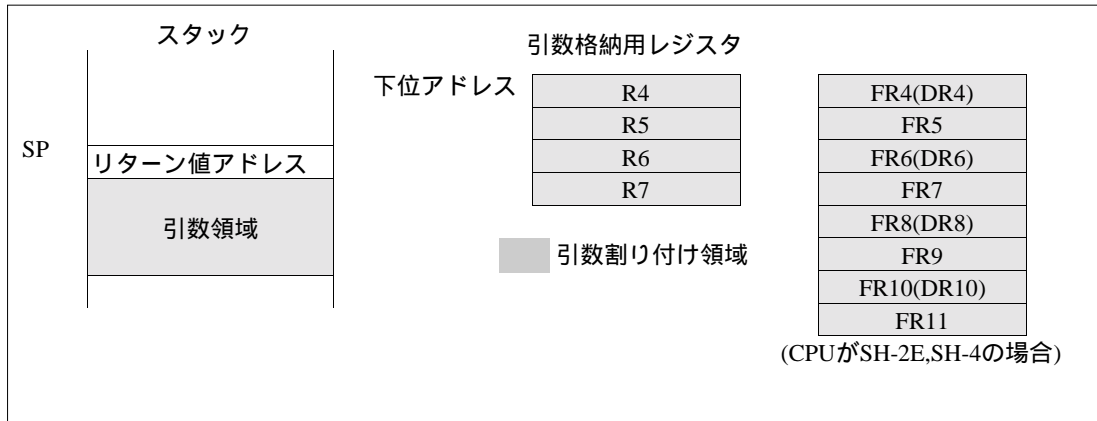


図 9.9 引数の割り付け領域

表 9.6 引数割り付け領域の一般規則

割り付け規則		
引数格納用レジスタ	レジスタで渡される引数 対象の型	スタックで渡される引数
R4 ~ R7	char, unsigned char, bool, short, unsigned short, int, unsigned int, long, unsigned long, float(CPU が SH-1、SH-2、SH-3 の場合),ポインタ,データ メンバへのポインタ,リファレンス	(1) 引数の型がレジスタ渡しの対象 の型以外のもの (2) プロトタイプ宣言により可変個 の引数を持つ関数として宣言してい るもの ^{*3} (3) 他の引数がすでに R4 ~ R7 に割 り付いている場合
FR4 ~ FR11 ^{*1}	SH-2E のとき ・引数が float 型 ・引数が double 型かつ double=float 指定 SH-4 のとき ・引数型が float 型かつ fpu=double オプション指定なし ・引数型が double 型かつ fpu=single 指定	(4) 他の引数がすでに FR4(DR4) ~ FR11(DR10)に割り付いている場合
DR4 ~ DR10 ^{*2}	SH-4 のとき ・引数型が double 型かつ fpu=single 指定なし ・引数型が float 型かつ fpu=double 指定	

【注】 *1 SH-2E、SH-4 の単精度浮動小数点用のレジスタです。

*2 SH-4 の倍精度浮動小数点用レジスタです。

*3 原型宣言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に割り付けます。

9. プログラミング

例

```
int f2(int,int,int,int,...);
:
f2(a,b,c,x,y,z);    x、y、z はスタックに割り付けます。
```

(c) 引数の割り付け

- 引数格納用レジスタへの割り付け

引数格納用レジスタには、ソースプログラムの宣言順に番号の小さいレジスタから割り付けます。引数格納用レジスタの割り付け例を図 9.10 に示します。

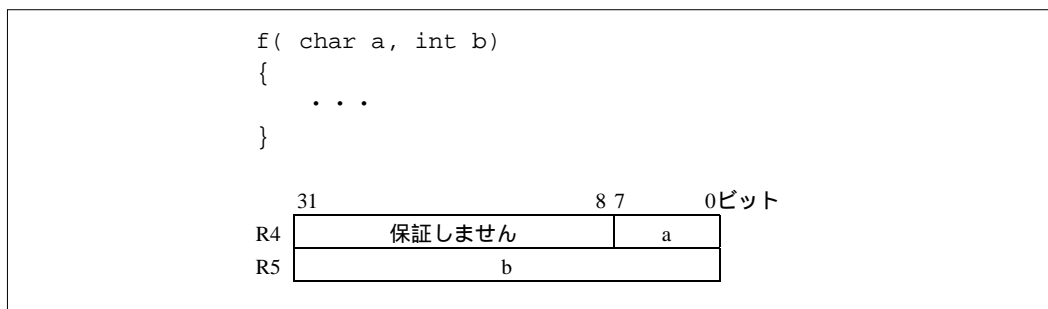


図 9.10 引数格納用レジスタの割り付け例

- スタック上の引数領域への割り付け

スタック上の引数領域には、ソースプログラム上で宣言した順に下位アドレスから割り付けます。

【注】 構造体、共用体、クラス型の引数に関する注意

構造体、共用体、クラス型の引数を設定する場合は、その型の本来の境界調整にかかわらず 4 バイト境界に割り付けられ、しかもその領域として 4 の倍数バイトの領域が使用されます。これは、SuperH RISC engine マイコンのスタックポインタが 4 バイト単位で変化するためです。

「9.3.3 引数割り付けの具体例」に、引数割り付けの具体例がありますので、あわせて参照してください。

(d) 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 9.7 を参照してください。

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、そのアドレスをリターン値アドレスに設定してから関数を呼び出します(図 9.11 参照)。関数のリターン値が void 型の場合、リターン値を設定しません。

表 9.7 リターン値の型と設定場所

リターン値の型	リターン値の設定場所
1 (signed) char, unsigned char, (signed) short, unsigned short, (signed) int, unsigned int, long, unsigned long float, ポインタ, bool リファレンス、データメンバへのポインタ	R0 : 32 ビット (signed) char, unsigned char の上位 3 バイト、(signed) short, unsigned short の上位 2 バイトの内容は保証しません。ただし、rtnext オプションを指定した場合は(signed) char, (signed) short 型は符号拡張、unsigned char, unsigned short 型はゼロ拡張を行います。 FR0 : 32 ビット (1)SH-2E のとき ・リターン値が float 型 ・リターン値が double 型かつ double=float 指定 (2)SH-4 のとき ・リターン値が float 型かつ fpu=double 指定なし ・リターン値が浮動小数点型かつ fpu=single 指定
2 double, long double 構造体、共用体、クラス型、 関数メンバへのポインタ	リターン値設定領域(メモリ) DR0:64 ビット SH-4 のとき ・リターン値が double 型かつ fpu=single 指定なし ・リターン値が浮動小数点型かつ fpu=double 指定

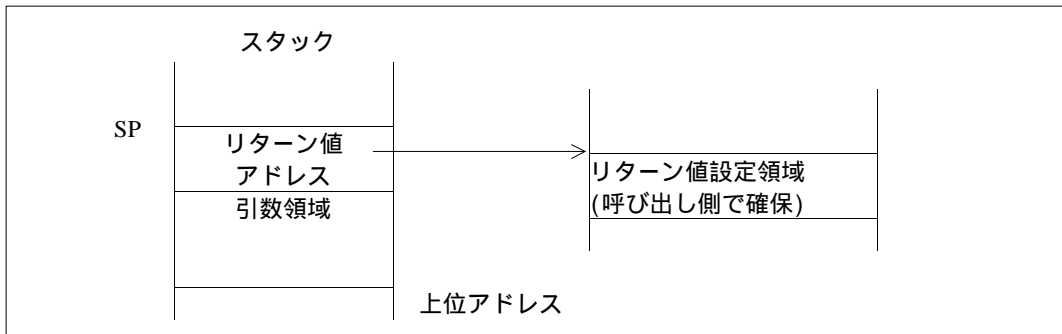


図 9.11 リターン値をメモリに設定する場合のリターン値の設定領域

9. プログラミング

9.3.3 引数割り付けの具体例

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 R4 ~ R7 に割り付けます。

R4	保証しない	1
R5	保証しない	2
R6		3
R7		4.0

例2 . レジスタに割り付けることができなかった引数は、スタックに割り付けます。また、引数の型が (unsigned)char 型、または、(unsigned)short 型でスタック上の引数領域に割り付く場合、4 バイトに拡張して割り付けます。

R4		1
R5	保証しない	2
R6		3
R7		4.0
		下位アドレス
引数領域	保証しない	5
(スタック)		上位アドレス

例3 . レジスタに割り付けられない型の引数は、スタックに割り付けます。

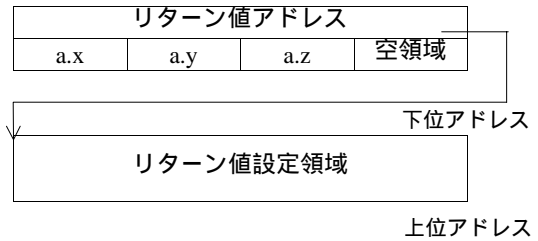
R4		1
R5		3
		下位アドレス
引数領域	a.x	
(スタック)	a.y	
		上位アドレス

例4 . 原型宣言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、宣言順にスタックに割り付けます。

R4		2
		下位アドレス
引数領域	1.0	
(スタック)	3	
		4
		上位アドレス

例5 . 関数の返す型が4バイトをこえる場合またはクラスの場合、引数領域の直前にリターン値アドレスを設定します。また、クラスのサイズが4の倍数バイトでないとき、空領域が生じます。

```
struct s{char x,y,z;}a;
double f(struct s);
:
f(a);
```

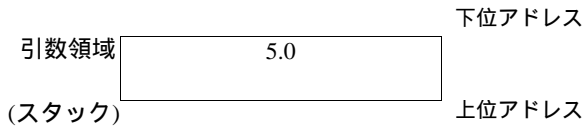


例6 . CPU が SH-2E の場合、float 型の引数は FPU レジスタに割り付きます。

```
int f(char,float,short,float,double);
:
f(1,2.0,3,4.0,5.0);
```

R4	保証しない	1
R5	保証しない	3
R6		
R7		

FR4	2.0
FR5	4.0
FR6	
FR7	
FR8	
FR9	
FR10	
FR11	



例7 . CPU が SH-4 の場合、float/double 型の引数は FPU レジスタに割り付きます。

```
int f(char,float,double,float,short);
:
f(1,2.0,4.0,5.0,3);
```

R4	保証しない	1
R5	保証しない	3
R6		
R7		

FR4(DR4)	2.0
FR5	5.0
FR6(DR6)	4.0
FR7	
FR8(DR8)	
FR9	
FR10(DR10)	
FR11	

9.3.4 レジスタとスタック領域の使用法

コンパイラのレジスタ、スタック領域の使用法を示します。

関数内でのレジスタ、スタック領域はすべてコンパイラが操作しますので、ユーザが特にこの領域の使用法に留意する必要はありません。

レジスタとスタック領域の使用法を図 9.12 に示します。

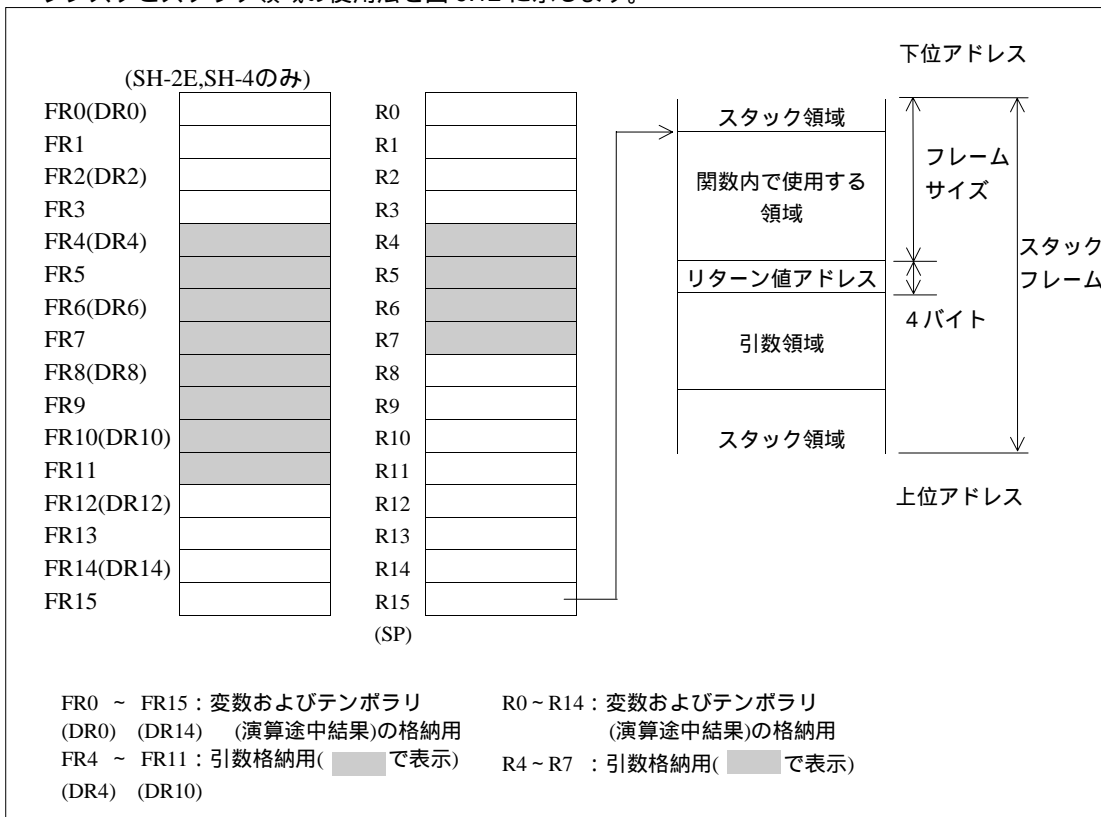


図 9.12 レジスタとスタック領域の使用法

9.4 プログラム作成上の注意事項

本節では、コンパイラにおけるコーディング上の注意事項と、コンパイルからデバッグまでのプログラム開発上の注意事項を述べます。

9.4.1 コーディング上の注意事項

(1) float 型パラメタをもつ関数

float 型のパラメタを受け渡している関数は、必ず原型宣言を行うか、float 型を double 型に変更してください。原型宣言のない float 型パラメタの受け渡しによるデータの値は保証されません。

例

```
void f(float);
void g()
{
    float a;
    ...
    f(a);
}
void f(float x)
{...}
```

関数 f は、float 型のパラメタをもつ関数です。この場合、必ず原型宣言を行ってください。

(2) C/C++ 言語で評価順序を規定していない式

C/C++ 言語で評価順序を規定していない式で、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、その動作は保証しません。

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。

(3) オーバフロー演算、ゼロ除算

オーバ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;
```

9. プログラミング

```
/* 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=999999999999; /* (W) 定数のオーバーフローを検出します */
fa=3.5e+40f; /* (W) 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 */
}
```

(4) 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中で、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 */
```

(5) 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 -)における絶対誤差	倍精度 2^{-39} (= 2^{-33})
	単精度 2^{-21} (= 2^{-19})
asin(1.0 -)における絶対誤差	倍精度 2^{-39} (= 2^{-28})
	単精度 2^{-21} (= 2^{-16})

9.4.2 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) 関数の原型宣言

関数を使用する前に原型宣言が必要です。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; // 内部的
```

```
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
}
```

9.4.3 プログラム開発上の注意事項

プログラムの作成からデバッグまでのプログラム開発上の注意事項を示します。

(1) CPU の選択に関する注意事項

(a) コンパイル、アセンブル時に指定する CPU 種別は統一してください

コンパイル、アセンブル時に `cpu` オプションで指定する CPU 種別は、必ず統一してください。異なる CPU 種別で作成したオブジェクトプログラムと一緒にリンクした場合、オブジェクトプログラム実行時の動作は保証しません。

(b) アセンブル時はコンパイル時の CPU と同じ CPU 種別を指定してください

コンパイラが生成したアセンブリプログラムをアセンブルするとき、コンパイル時と同じ CPU 種別を `cpu` オプションで指定してください。

(c) リンク時には CPU 種別に合わせた標準ライブラリを指定してください。

CPU 種別に対応するライブラリを指定してください。それ以外のライブラリを指定した場合の動作は保証しません。

(2) オプションに関する注意事項

以下のオプションは、コンパイル時、ライブラリ構築時に必ず統一して下さい。異なるオプションを用いて作成したオブジェクトプログラムと一緒にリンクした場合、オブジェクトプログラム実行時の動作は保証しません。

- `endian = big | little` (SH-3,SH-4)
- `pic = 0 | 1` (SH-1 以外)
- `fpu = single | double` (SH-4)
- `fpscr = safe | aggressive` (SH-4)
- `round = zero | nearest` (SH-4)
- `denormalize = on | off` (SH-4)
- `double = float` (SH-4 以外)
- `exception | noexception`
- `rtti = on | off`

10. C/C++言語仕様

10.1 言語仕様

10.1.1 コンパイラの仕様

言語仕様で規定していない処理系定義項目について、コンパイラの仕様を示します。

(1) 環境

表 10.1 環境の仕様

項目	コンパイラの仕様
1 main 関数への実引数の意味	規定しません。
2 対話的入出力装置の構成	規定しません。

(2) 識別子

表 10.2 識別子の仕様

項目	コンパイラの仕様
1 外部結合とならない識別子(内部名)の有効文字数	8189 文字まで有効です。
2 外部結合となる識別子(外部名)の有効文字数	8191 文字まで有効です。
3 外部結合となる識別子(外部名)の大文字と小文字の区別	大文字と小文字を区別します。

(3) 文字

表 10.3 文字の仕様

項目	コンパイラの仕様
1 ソース文字集合および実行環境文字集合の要素	どちらも ASCII 文字集合です。ただし、文字列、文字定数にはシフト JIS、EUC 漢字コードまたは Latin1 コードを記述できます。
2 多バイト文字のコード化で使用されるシフト状態	シフト状態はサポートしていません。
3 プログラム実行時の文字集合の文字のビット数	ビット数は 8 ビットです。
4 文字定数内、文字列内のソース文字集合の文字と実行環境文字集合の文字との対応付け	同じ ASCII 文字に対応します。
5 言語で規定していない文字や拡張表記を含む整数文字定数の値	言語で規定する以外の文字、拡張表記はサポートしていません。
6 2 文字以上の文字を含む文字定数または 2 文字以上の多バイト文字を含む広角文字定数の値	文字定数は上位 2 文字を有効とします。広角文字定数はサポートしていません。また、1 文字より多く指定した場合はウォーニングエラーを出力します。
7 多バイト文字を広角文字に変換するために使用される locale の仕様	locale はサポートしていません。
8 char 型の値	signed char 型と同じ値の範囲を持ちます。

10. C/C++言語仕様

(4) 整数

表 10.4 整数の仕様

項目	コンパイラの仕様
1 整数型の表現方法とその値	表 10.5 に示します。
2 整数の値がより短いサイズの符号付き整数型、または符号なし整数型を同一のサイズの符号付き整数型に変換したときの値(結果の値が変換先の型で表現できない場合)	整数の値の下位2バイトあるいは下位1バイトが変換後の値になります。
3 符号付き整数に対するビットごとの演算の結果	符号付きの値になります。
4 整数除算における剰余の符号	被除数の符号と同符号になります。
5 負の値を持つ符号付き汎整数型の右シフトの結果	符号ビットを保持します。

表 10.5 整数型とその値の範囲

型	値の範囲	データサイズ
1 char	-128 ~ 127	1 バイト
2 signed char	-128 ~ 127	1 バイト
3 unsigned char	0 ~ 255	1 バイト
4 short	-32768 ~ 32767	2 バイト
5 unsigned short	0 ~ 65535	2 バイト
6 int	-2147483648 ~ 2147483647	4 バイト
7 unsigned int	0 ~ 4294967295	4 バイト
8 long	-2147483648 ~ 2147483647	4 バイト
9 unsigned long	0 ~ 4294967295	4 バイト

(5) 浮動小数点

表 10.6 浮動小数点の仕様

項目	コンパイラの仕様
1 浮動小数点型の表現方法とその値	浮動小数点型には、float 型、double 型と long double 型があります。浮動小数点型の内部表現や変換仕様、演算仕様等の性質は「10.1.3 浮動小数点数の仕様」で説明します。表 10.7 に、浮動小数点型の表現可能な値の限界値を示します。
2 整数を本来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したときの切り捨て方向	
3 浮動小数点数をより狭い浮動小数点数に変換したときの切り捨てまたは丸め方法	

表 10.7 浮動小数点数の限界値

項目	限界値	
	10 進数表現 ¹⁾	16 進数表現
1 float 型の最大値	3.4028235677973364e+38f (3.4028234663852886e+38f)	7f7fffff
2 float 型の正の最小値	7.0064923216240862e-46f (1.4012984643248171e-45f)	00000001
3 double } 型の最大値	1.7976931348623158e+308 (1.7976931348623157e+308)	7feffffffffff
4 double } 型の正の long double } 最小値	4.9406564584124655e-324 (4.9406564584124654e-324)	0000000000000001

【注】 *1 10 進数表現の限界値は 0 または無限大にならない限界値です。また、()内は理論値を示します。

*2 double=float を指定した場合、double 型は float 型と同じ値となります。
 fpu=single を指定した場合、double、long double 型は float 型と同じ値になります。
 fpu=double を指定した場合、float 型は double 型と同じ値になります。

(6) 配列とポインタ

表 10.8 配列とポインタの仕様

項目	コンパイラの仕様
1 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 (size_t)	unsigned long 型
2 ポインタ型から整数型への変換 (ポインタ型のサイズ 整数型のサイズ)	ポインタ型の下位バイトの値になります。
3 ポインタ型から整数型への変換 (ポインタ型のサイズ < 整数型のサイズ)	符号拡張します。
4 整数型からポインタ型への変換 (整数型のサイズ ポインタ型のサイズ)	整数型の下位バイトの値となります。
5 整数型からポインタ型への変換 (整数型のサイズ < ポインタ型のサイズ)	符号拡張します。
6 同じ配列内のメンバのポインタ間の差を保持するために必要な整数の型 (ptrdiff_t)	int 型

(7) レジスタ

表 10.9 レジスタの仕様

項目	コンパイラの仕様
1 レジスタに割り付けることができるレジスタ変数の最大数	7 個 char, unsigned char, bool, short, unsigned short, int, unsigned int, long, unsigned long, ポインタ
	4 個 float ^{*1}
	2 個 double ^{*1}
2 レジスタに割り付けることができるレジスタ変数の型	char, unsigned char, bool, short, unsigned short, int, unsigned int, long, unsigned long, float ^{*1} , double ^{*1} , ポインタ

【注】 *1 CPU が SH-2E、または SH-4 の場合のみです。

10. C/C++言語仕様

(8) クラス、構造体、共用体、列挙型、ビットフィールド

表 10.10 クラス、構造体、共用体、列挙型、ビットフィールドの仕様

項目	コンパイラの仕様
1 異なる型のメンバでアクセスされる共用体型のメンバ参照	参照はできますが、値は保証しません。
2 クラス・構造体メンバの境界調整	クラス・構造体メンバ中のデータサイズの最大値が境界調整数になります。割り付け方の詳細な仕様は「10.1.2(2) 複合型、クラス型」を参照してください。
3 単なる int 型のビットフィールドの符号	signed int 型
4 int 型のサイズ内のビットフィールドの割り付け順序	上位ビットから割り付けます。
5 int 型のサイズ内にビットフィールドが割り付けられているとき、次に割り付けるビットフィールドのサイズが int 型内の残っているサイズを超えたときの割り付け方	次の int 型の領域に割り付けます。
6 ビットフィールドで許される型指定子	char, unsigned char, bool, short, unsigned short, int, unsigned int, long, unsigned long, enum
7 列挙型の値を表現する整数型	int 型

ビットフィールドの割り付け方の詳細については、「10.1.2(3) ビットフィールド」を参照してください。

(9) 修飾子

表 10.11 修飾子の仕様

項目	コンパイラの仕様
1 volatile 型データへのアクセスの種類	規定しません。

(10) 宣言

表 10.12 宣言の仕様

項目	コンパイラの仕様
1 基本型（算術型、構造体型、共用体型）を修飾する宣言子の数	16 個まで指定できます。

基本型を修飾する型の数の数え方を、以下に例を用いて示します。

例：

- (i) `int a;` aはint型(基本型)であり、基本型を修飾する型の数は0個です。
- (ii) `char *f();` fはchar型(基本型)へのポインタ型を返す関数型であり、基本型を修飾する型の数は2個です。

(11) 文

表 10.13 文の仕様

項目	コンパイラの仕様
1 一つの switch 文中で指定できる case ラベルの数	511 個まで指定できます。

(12) プリプロセッサ

表 10.14 プリプロセッサの仕様

	項 目	コンパイラの仕様
1	条件コンパイルの定数式内の単一文字の文字定数と実行環境文字集合の対応	プリプロセッサ文の文字定数と実行環境文字集合は一致します。
2	インクルードファイルの読み込み方法	「<」、「>」で囲まれたファイルは include オプションで指定されたパスから読み込みます。 ファイルが見つからない場合、環境変数 SHC_INC 指定ディレクトリ、システムディレクトリ(SHC_LIB)の順序で各ディレクトリを検索します。
3	二重引用符で囲まれたインクルードファイルのサポートの有無	サポートします。インクルードファイルを現在のディレクトリから読み込みます。現在のディレクトリになければ、本表 2 項の読み込み方法に従います。
4	ソースファイルの文字の並びの対応(マクロ展開後の文字列の空白文字)	空白文字列は、空白文字 1 文字として展開します。
5	#pragma 文の動作	「10.2.1 #pragma 拡張子」を参照してください。
6	__DATE__、__TIME__の値	コンパイル開始時のホストマシンのタイムに基づく値が設定されます。

10.1.2 データの内部表現

本節では、データ型と、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。
- データの境界調整数
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バイト境界調整、偶数バイトに割り付ける2バイト境界調整、4の倍数バイトに割り付ける4バイト境界調整があります。
- データの範囲
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
複合型(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 10.15 に示します。

表 10.15 スカラ型・基本型の内部表現

	データ型	サイズ (byte)	境界調整 数(byte)	符号の 有無	データの範囲	
					最小値	最大値
1	char	1	1	有	-2^7 (-128)	2^7-1 (127)
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	unsigned short	2	2	無	0	$2^{16}-1$ (65535)
6	int	4	4	有	-2^{31} (-2147483648)	$2^{31}-1$ (2147483647)
7	unsigned int	4	4	無	0	$2^{32}-1$ (4294967295)
8	long	4	4	有	-2^{31} (-2147483648)	$2^{31}-1$ (2147483647)
9	unsigned long	4	4	無	0	$2^{32}-1$ (4294967295)
10	enum	4	4	有	-2^{31} (-2147483648)	$2^{31}-1$ (2147483647)
11	float	$4^{1,3}$	4	有	-	+
12	double long double	$8^{1,3}$	4	有	-	+
13	ポインタ	4	4	無	0	$2^{32}-1$ (4294967295)
14	bool ²	4	4	有	-	-
15	リファレンス ²	4	4	無	0	$2^{32}-1$ (4294967295)
16	データメンバへのポインタ ²	4	4	有	0	$2^{32}-1$ (4294967295)
17	関数メンバへのポインタ ^{2,4}	12	4	-	-	-

- 【注】 *1 double=floatを指定した場合、double型のサイズは4バイトになります。
 *2 C++コンパイルのみ有効です。
 *3 cpu=sh4かつfpu=singleを指定した場合、double,long double型を4バイト(float型)として扱います。
 また、cpu=sh4かつfpu=doubleを指定した場合、float型を8バイト(double型)として扱います。
 *4 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
    };
};
```

(2) 複合型(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 10.16 に複合型、クラス型の内部表現を示します。

表 10.16 複合型、クラス型の内部表現

データ型	境界調整数(byte)	サイズ(byte)	データの割り付け例
1 配列型	配列要素の境界調整数	配列要素の数×要素サイズ	char a[10]; 境界調整数 1byte サイズ 10byte
2 構造体型	構造体メンバの境界調整数のうち最大値	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 { 境界調整数 1byte char a,b; サイズ 2byte };
3 共用体型	共用体メンバの境界調整数のうち最大値	メンバのサイズの最大値 「(b) 共用体データの割り付け方」参照	union { 境界調整数 1byte char a,b; サイズ 1byte };
4 クラス型	1)仮想関数がある場合: 常に4 2)上記以外: データメンバの境界調整数のうち最大値	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{ 境界調整数 4byte virtual void f(); サイズ 8byte }; class A{ 境界調整数 1byte char a; サイズ 1byte };

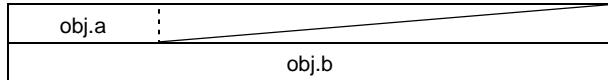
以下の例でサイズを明記していない は、4バイトを表します。 は境界調整領域を表します。

(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、そのメンバのデータ型の境界調整数に合わせるために直前のメンバとの間に空き領域が生じる場合があります。

例：

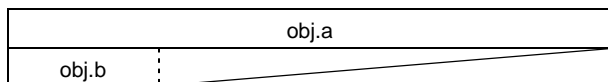
```
struct {
    char a;
    int b;
} obj;
```



- 構造体が4バイトの境界調整数を持ち、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;
```

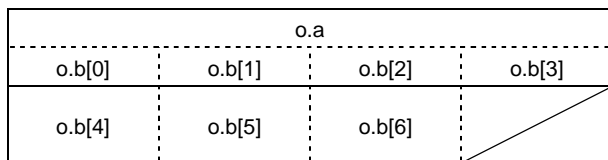


(b) 共用体データの割り付け方

- 共用体が4バイトの境界調整数を持ち、メンバのサイズの最大値が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;
```

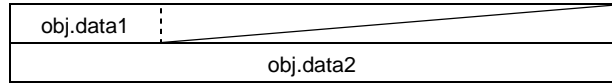


(c) クラスデータの割り付け方

- 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：

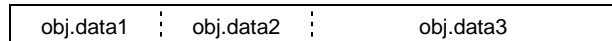
```
class A{
    char data1;
    int data2;
public:
    A();
    int getData1(){return data1;}
}obj;
```



- 境界調整数が1の基底クラスから派生したクラスの場合、先頭メンバが1byteデータの場合、空き領域を作らないようにデータメンバを割り付けます。

例：

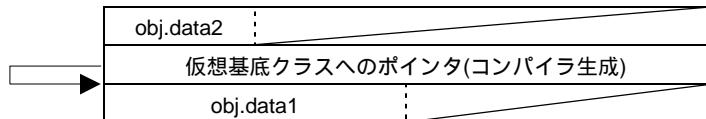
```
class A{
    char data1;
};
class B:public A{
    char data2;
    short data3;
}obj;
```



- 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：

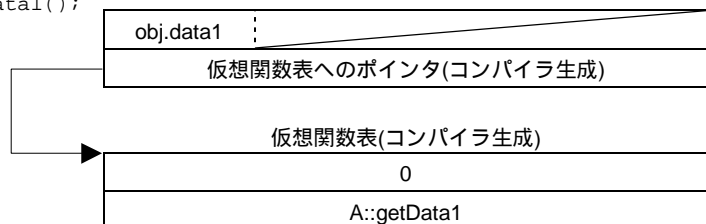
```
class A{
    short data1;
};
class B: virtual protected A{
    char data2;
}obj;
```



- 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：

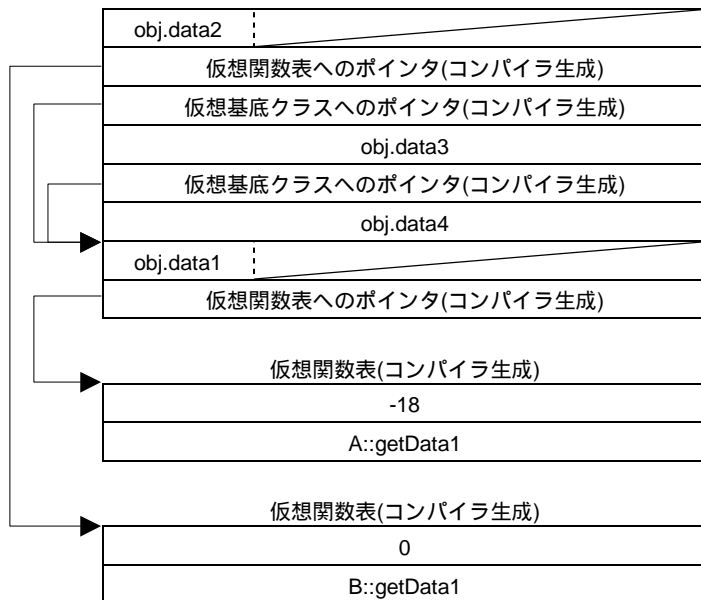
```
class A{
    char data1;
public:
    virtual int getData1();
}obj;
```



- ・ 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：

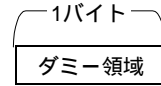
```
class A{
    char data1 ;
    virtual short getData1();
};
class B:virtual public A{
    char data2;
    char getData2();
    short getData1();
};
class C:virtual protected A{
    int data3;
};
class D:virtual public A,public B,public C{
public:
    int data4;
    short getData1();
}obj;
```



- 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：

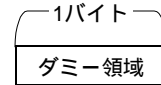
```
class A{
    void fun();
}obj;
```



- 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。

例：

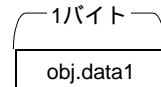
```
class A{
    void fun();
};
class B: A{
    void sub();
}obj;
```



- 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、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 10.17 にビットフィールドメンバの仕様を示します。

表 10.17 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	(signed)char、unsigned char、bool ¹ 、 (signed)short、unsigned short、enum、 (signed)int、unsigned int、 (signed)long、unsigned 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 を指定した型) ゼロ拡張 ³ 符号あり(unsigned を指定しない型) 符号拡張 ⁴

【注】 *1 C++プログラムのみ bool を指定できます。

*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。0 と 1 を表現する場合には、必ず符号なし(unsigned)で宣言してください。

*3 ゼロ拡張：拡張するとき上位のビットにゼロを補います。

*4 符号拡張：拡張するとき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。

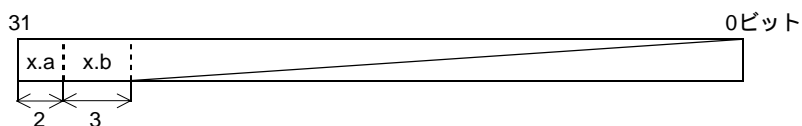
(b) ビットフィールドの割り付け方

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左(上位ビット側)から順に詰め込みます。

例：

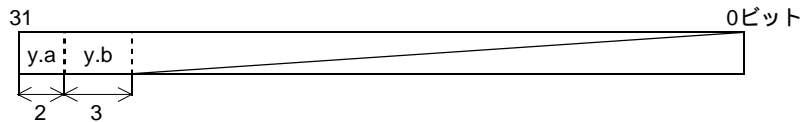
```
struct b1 {
    int a:2;
    int b:3;
} x;
```



- 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：

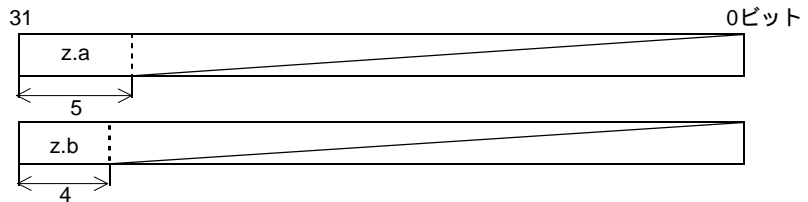
```
struct b1 {
    long          a:2;
    unsigned int  b:3;
} y;
```



- 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：

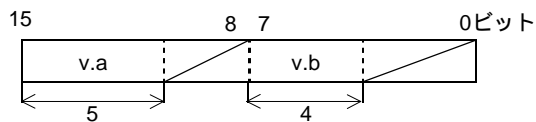
```
struct b1 {
    int   a:5;
    char  b:4;
} z;
```



- 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：

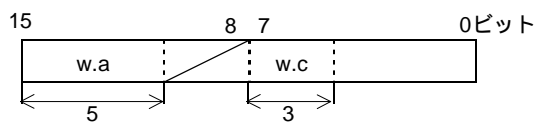
```
struct b2 {
    char a:5;
    char b:4;
} v;
```



- 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。

例：

```
struct b2 {
    char a:5;
    char :0;
    char c:3;
} w;
```



(4) Little Endian のメモリ割り付け

Little 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((signed) char、unsigned char、bool 型)

1 バイトデータの中のビット並び順は、Big Endian の場合も、Little Endian の場合も同じです。

(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、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

Big Endian:	0x100番地:	0x12	Little Endian:	0x100番地:	0x34
	0x101番地:	0x34		0x101番地:	0x12

(c) 4 バイトデータ((signed) int、unsigned int、(signed) long、unsigned long、float 型)

4 バイトデータの中のバイト並び順は、Big Endian と Little Endian で 4 バイトのデータの順序が逆になります。

例

0x100番地に4バイトデータ0x12345678がある場合

Big Endian:	0x100番地:	0x12	Little Endian:	0x100番地:	0x78
	0x101番地:	0x34		0x101番地:	0x56
	0x102番地:	0x56		0x102番地:	0x34
	0x103番地:	0x78		0x103番地:	0x12

(d) 8 バイトデータ(double 型)

8 バイトデータの中のバイト並び順は、Big Endian と Little 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

Big Endian:	0x100番地:	0x01	Little Endian:	0x100番地:	0xef
	0x101番地:	0x23		0x101番地:	0xcd
	0x102番地:	0x45		0x102番地:	0xab
	0x103番地:	0x67		0x103番地:	0x89
	0x104番地:	0x89		0x104番地:	0x67
	0x105番地:	0xab		0x105番地:	0x45
	0x106番地:	0xcd		0x106番地:	0x23
	0x107番地:	0xef		0x107番地:	0x01

(e) 複合型、クラス型データ

複合型、クラス型データの各メンバの割り付けは Big 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

Big Endian:	0x100番地: 0x12	Little Endian:	0x100番地: 0x34
	0x101番地: 0x34		0x101番地: 0x12
	0x102番地: 空き領域		0x102番地: 空き領域
	0x103番地: 空き領域		0x103番地: 空き領域
	0x104番地: 0x56		0x104番地: 0xbc
	0x105番地: 0x78		0x105番地: 0x9a
	0x106番地: 0x9a		0x106番地: 0x78
	0x107番地: 0xbc		0x107番地: 0x56

(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 Big 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

Big Endian:	0x100番地: 0x00	Little Endian:	0x100番地: 0x01
	0x101番地: 0x01		0x101番地: 0x00
	0x102番地: 0x00		0x102番地: 0x02
	0x103番地: 0x02		0x103番地: 0x00
	0x104番地: 0x08		0x104番地: 0x00
	0x105番地: 0x00		0x105番地: 0x08
	0x106番地: 空き領域		0x106番地: 空き領域
	0x107番地: 空き領域		0x107番地: 空き領域

10.1.3 浮動小数点数の仕様

(1) 浮動小数点数の内部表現

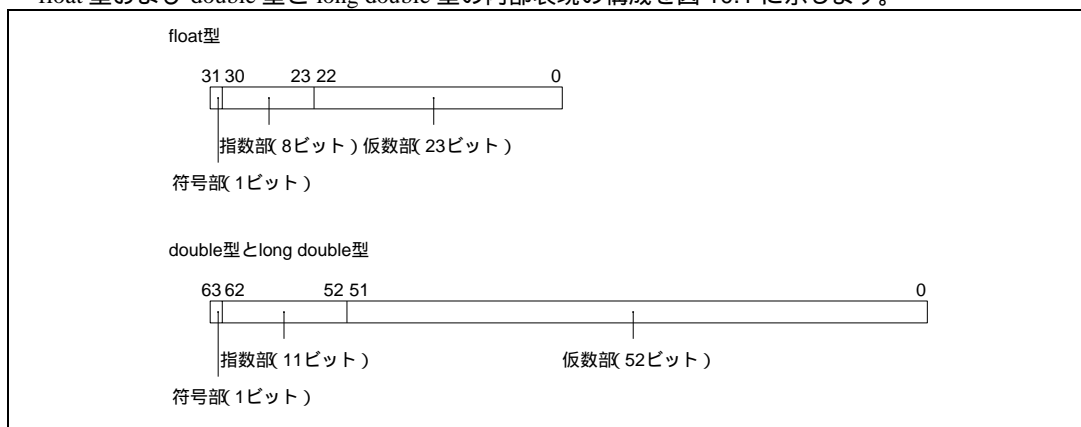
コンパイラで扱う浮動小数点数の内部表現は、IEEE の標準形式に従っています。ここでは、IEEE 形式の浮動小数点数の内部表現の概要について述べます。

(a) 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。

(b) 内部表現の構成

float 型および double 型と long double 型の内部表現の構成を図 10.1 に示します。



- 【注】 double=float を指定した場合、double 型は float 型と同じ内部表現となります。
 cpu=sh4 かつ fpu=single を指定した場合、double 型、long double 型は float 型と同じ内部表現となります。
 cpu=sh4 かつ fpu=double を指定した場合、float 型は double 型と同じ内部表現になります。

図 10.1 浮動小数点数の内部表現の構成

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
浮動小数点数の符号を示します。0のとき正、1のとき負を示します。
 - (ii) 指数部
浮動小数点数の指数を2のべき乗で示します。
 - (iii) 仮数部
浮動小数点数の有効数字に対応するデータです。
- (c) 表現する値の種類
- 浮動小数点数は、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。浮動小数点数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。

浮動小数点数の表現する値を決定する条件を表 10.18 に示します。

表 10.18 浮動小数点数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数を表現しますが、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。

cpu=sh4 かつ denormalize=off を指定した場合、非正規化数は0として扱います。

cpu=sh4 かつ 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。

(2) 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、 $1 \sim 254(2^8-2)$ 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、 $0 \sim 2^{23}-1$ 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、

$$(-1)^{\langle \text{符号部} \rangle} \times 2^{\langle \text{指数部} \rangle - 127} \times (1 + \langle \text{仮数部} \rangle \times 2^{-23})$$

となります。

例：

$$\begin{array}{ccccccc} 31 & 30 & & 23 & 22 & & 0 \\ \hline 1 & 10000000 & 1 & 100000000000000000000000 & 00000000 & & 0 \end{array}$$

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、 $1 \sim 2^{23}-1$ 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、

$$(-1)^{\langle \text{符号部} \rangle} \times 2^{-126} \times (\langle \text{仮数部} \rangle \times 2^{-23})$$

となります。

例：

31 30	23 22	0
0	110000000000000000000000	00000000

符号： +

指数： - 126

仮数： 0.11₍₂₎ = 0.75

値： 0.75 × 2⁻¹²⁶

(2)は2進数を意味します。

(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。ゼロの符号による、各演算での機能の違いについては「10.1.3(4) 浮動小数点演算の仕様」を参照してください。

(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(2⁸-1)です。

仮数部は0です。

(v) 非数

指数部は255(2⁸-1)です。

仮数部は0以外の値です。

【注】 CPU が SH-2E、SH-4 の場合、仮数部の最上位ビットが 0 の非数を qNaN、仮数部の最上位ビットが 1 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。

(3) 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(2¹¹-2)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 ~ 2⁵²-1の値となります。実際の仮数は、2⁵²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、

$$(-1)^{\langle \text{符号部} \rangle} \times 2^{\langle \text{指数部} \rangle - 1023} \times (1 + \langle \text{仮数部} \rangle \times 2^{-52})$$

となります。

例：

63	62	52	51	0
0 0111111111				111000

符号： +

指数： $1111111111_{(2)} - 1023 = 0$ 仮数： $1.111_{(2)} = 1.875$ (2) は2進数を意味します。値： $1.875 \times 2^0 = 1.875$

(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、 $1 \sim 2^{52}-1$ 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、

$$(-1)^{\langle \text{符号部} \rangle} \times 2^{-1022} \times (\langle \text{仮数部} \rangle \times 2^{-52})$$

となります。

例：

63	62	52	51	0
1 0000000000				111000

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。ゼロの符号による、各演算での機能の違いについては「10.1.3(4) 浮動小数点演算の仕様」を参照してください。

(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。

【注】 CPUがSH-2E、SH-4の場合、仮数部の最上位ビットが0の非数をqNaN、仮数部の最上位ビットが1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) 浮動小数点演算の仕様

本項では、C/C++言語の機能として表現されている浮動小数点の四則演算、およびコンパイル時やCライブラリ関数の処理で生じる浮動小数点の10進表現と内部表現の間の変換の仕様について解説します。

(a) 四則演算の仕様

(i) 結果の値の丸め方

浮動小数点の四則演算の結果の正確な値が、内部表現の仮数の有効数字を超えた場合は、以下の規則に従って丸めを行います。

- [1] 結果の値は、その値を近似する二つの浮動小数点数の内部表現のうち、近い方に向かって丸めます。
- [2] 結果の値が、その値を近似する二つの浮動小数点数のちょうど中央になる場合は、仮数の最後の桁が0となる方向に丸めます。
- [3] CPUがSH-2Eの場合、有効数字を超える部分を切り捨てます。
- [4] CPUがSH-4の場合、round=nearestを指定した場合、有効数字を超える部分を四捨五入し、round=zeroを指定した場合、有効数字を超える部分を切り捨てます。

(ii) オーバフロー、アンダフロー、無効演算時の処理

実行時のオーバフロー、アンダフロー、無効演算に対しては、以下の処理を行います。

- [1] オーバフローの場合は、結果の符号に従って正または負の無限大になります。
- [2] アンダフローの場合は、結果の符号に従って正または負のゼロになります。
- [3] 無効演算は、符号が逆の無限大を加算した場合、符号が同じ無限大を減算した場合、ゼロと無限大を乗算した場合、ゼロをゼロで、あるいは無限大を無限大で除算した場合に生じます。
これらの場合、結果は非数になります。
- [4] 浮動小数点数から整数へ変換したときにオーバフローが生じた場合、結果の値は保障されません。

【注】 定数式に関しては、コンパイル時に演算を行います。この時にオーバフロー、アンダフロー、無効演算を検出した場合は、ウォーニングレベルのエラーになります。

(iii) 特殊な値(ゼロ、無限大、非数)の演算に関する注意事項

- [1] 正のゼロと負のゼロの和は正のゼロとなります。
- [2] 同符号のゼロの差は正のゼロとなります。
- [3] 被演算子の一方あるいは両方に非数を含む演算の結果は、常に非数になります。
- [4] 比較演算においては、正のゼロと負のゼロは等しいものとして扱います。
- [5] 被演算子の一方あるいは両方が非数であるような比較演算、等値演算の結果は、「!=」については常に真、その他は常に偽となります。

(b) 10進表現と内部表現の間の変換

本項ではソースプログラム上の浮動小数点定数と内部表現の間の変換、あるいはCライブラリ関数によるASCII文字列による浮動小数点数の10進表現と、内部表現の間の変換の仕様について解説します。

(i) 10進表現から内部表現に変換する場合、まず10進表現を10進表現の正規形に変換します。

10進表現の正規形は、「 $\pm M \times 10^N$ 」の形式で、M、Nの範囲は以下のとおりです。

- [1] float型の正規形

0	M	10^0	-1
0	N	99	

[2] double型とlong double型の正規形

0 M $10^{17}-1$
0 N 999

正規形に変換できない10進表現については、オーバフロー、またはアンダフローになります。また、10進表現が正規形よりも多くの有効数字を含んでいる場合は、下位の桁は切り捨てます。これらの場合、コンパイル時にはウォーニングレベルのエラーになり、実行時には対応するエラーの番号を変数errnoに設定します。

また、正規形に変換するためには、もとの10進表現のASCII文字列としての長さが511文字以下でなければなりません。そうでない場合、コンパイル時にはエラーになり、実行時には対応するエラーの番号を変数errnoに設定します。

内部表現から10進表現に変換する場合には、一度10進表現の正規形に変換してから、指定した書式に従ってASCII文字列に変換します。

(ii) 10進表現の正規形と内部表現の間の変換

10進表現の正規形と内部表現の間の変換は、指数が大きいときや小さいときには、正確な変換ができません。以下に、正確な変換ができる範囲と、その範囲外の場合での誤差の限界値について解説します。

[1] 正確な変換ができる範囲

以下に示す指数の範囲の浮動小数点数については、「(a)(i) 結果の値の丸め方」に示す丸めが正確に行われます。この範囲ではオーバフロー、アンダフローは生じません。

float型の場合： 0 M 10^9-1 、0 N 13

double型とlong double型の場合： 0 M $10^{17}-1$ 、0 N 27

[2] 誤差の限界値

[1]で示す範囲に入っていない値を変換する場合の誤差と、正確な丸めを行ったときの誤差の差は、有効数字の最小位桁の0.47倍を超えません。

また、[1]で示した範囲を超えている場合、変換の際にオーバフローやアンダフローが生じる場合があります。この場合、コンパイル時にはウォーニングレベルのエラーになり、実行時には対応するエラーの番号を変数 errno に設定します。

10.1.4 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10.19に示します。

表 10.19 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	乗法式
5	+ -	左	加法式
6	<< >>	左	シフト式
7	< <= > >=	左	関係式
8	== !=	左	等価式
9	&	左	ビット毎の AND 式
10	^	左	ビット毎の XOR 式
11		左	ビット毎の OR 式
12	&&	左	論理 AND 演算
13		左	論理 OR 式
14	?:	左	条件式
15	= += == *= /= %= <<= >>= &= = ^=	右	代入式
16	,	左	コンマ式

10.2 拡張機能

コンパイラの拡張機能として、以下の機能をサポートしています。

- #pragma 拡張子
- セクションアドレス演算子
- 組み込み関数

10.2.1 #pragma 拡張子

#pragma 拡張子の一覧を示します。

表 10.20 メモリ配置に関する拡張機能

#pragma 拡張子	機能
1 #pragma section	セクションの切り替え指定
2 #pragma abs16	2 バイトアドレスの指定
3 #pragma stacksize	スタックセクションの作成

表 10.21 関数に関する拡張機能

#pragma 拡張子	機能
1 #pragma interrupt	割り込み関数の作成
2 #pragma inline	関数のインライン展開を指定
3 #pragma inline_asm	アセンブリ記述関数のインライン展開
4 #pragma regsave #pragma noregsave	レジスタの退避 / 回復コード出力の制御
#pragma noregalloc	
5 #pragma entry	エントリ関数の作成

表 10.22 その他の拡張機能

#pragma 拡張子	機能
1 #pragma global_register	グローバル変数のレジスタ割り付け
2 #pragma gbr_base #pragma gbr_base1	GBR ベース変数の指定

上記拡張機能には、データメンバ、関数メンバが指定可能な機能があります。指定方法は(クラス名::メンバ名)です。指定可能なメンバの種類は、各機能の記述方法を参照してください。

(1) メモリ配置に関する拡張機能

セクション切り替え指定

#pragma section

書 式 #pragma section [{<名前>|<数値>}]

説 明 コンパイラの出力するセクション名を切り替えます。
デフォルト、切り替え後のセクション名は表 10.23 のとおりです。

表 10.23 セクション切り替え機能とセクション名

対象領域	指定方法	デフォルト名	切り替え後
1 プログラム領域		P ¹	P<xx>
2 定数領域	#pragma section <xx>	C ¹	C<xx>
3 初期化データ領域		D ¹	D<xx>
4 未初期化データ領域		B ¹	B<xx>

【注】*1 section オプションでデフォルトセクション名を変更できます。
<名前>や<数値>を省略すると、以降はデフォルトのセクション名になります。

```
例
#pragma section abc
int a;          /* a はセクション Babc に割り付きます */
const int c=1; /* c はセクション Cabc に割り付きます */
void f(void)   /* f はセクション Pabc に割り付きます */
{
    a=c;
}
#pragma section
int b;          /* b はセクション B に割り付きます */
void g(void)   /* g はセクション P に割り付きます */
{
    b=c;
}
```

備 考 #pragma section は関数定義の外で宣言しなければなりません。
1 ファイルで宣言できるセクション名は最大 64 個です。

2 バイトアドレス指定

#pragma abs16

- 書式** `#pragma abs16 (<識別子>[,...])`
- 説明** H'00000000 ~ H'00007FFF 番地および H'FFFF8000 ~ H'FFFFFFFF 番地に配置した変数または関数のアドレスを 2 バイトの値として扱います。これによってプログラムサイズを削減できます。
識別子には、変数、グローバル関数、静的データメンバ及び関数メンバを指定できます。
- 例**
- ```
#pragma abs16(x, y, z)
extern int x();
int y;
long z;
void f(void){
 z=x()+y;
}
```
- 備考**     **#pragma abs16** は、自動オブジェクトや非静的データメンバを指定できません。  
**#pragma abs16** で宣言された変数および関数は、必ず H'00000000 ~ H'00007FFF 番地または、H'FFFF8000 ~ H'FFFFFFFF 番地に配置してください。

## スタックセクション作成

***#pragma stacksize***

- 書式**     `#pragma stacksize <定数>`
- 説明**     セクション名 S、サイズ<定数>のスタックセクションを作成します。
- 例**
- ```
#pragma stacksize 100
```
- <コード展開例>
- ```
.SECTION S, STACK
.RES.B 100
```
- 備考**     サイズ<定数>は必ず 4 の倍数を指定してください。  
**#pragma stacksize** はファイル内で一回しか指定できません。

## (2) 関数に関する拡張機能

## 割り込み関数作成

**#pragma interrupt**

書 式 #pragma interrupt (<関数名>[(割り込み仕様)][,...])

説 明 #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 10.24 に示します。

表 10.24 割り込み仕様の一覧

| 項目             | 形式   | オプション                                            | 指定内容                                                                                  |
|----------------|------|--------------------------------------------------|---------------------------------------------------------------------------------------|
| 1 スタック切り替え指定   | sp = | { <変数>   &<変数>   <定数>   <変数>+<定数>   &<変数>+<定数> } | 新しいスタックのアドレスを変数または定数で指定<br><変数> : 変数(ポインタ型)<br>&<変数> : 変数(オブジェクト型)のアドレス<br><定数> : 定数値 |
| 2 トラップ命令リターン指定 | tn = | <定数>                                             | 終了を TRAPA 命令で指定<br>定数値(トラップベクタ番号)                                                     |

#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、トラップ命令リターンを指定した場合は TRAPA 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。また、スタック切り換え指定とトラップ命令リターン指定は重複して指定できます。

## 例

```
extern int STK[100];
#pragma interrupt(f(sp=STK+100, tn=10),A::g)
class A{
public:
 static void g();
};
```

## 説明

- (a) スタック切り替え指定  
STK+100 を割り込み関数「f」で使用するスタックポインタとして設定します。
- (b) トラップ命令リターン指定  
割り込み関数終了時に TRAPA #10 でトラップ例外処理を開始します。トラップ例外処理開始時の SP は図 10.2 のようになっています。トラップルーチンの側で RTE 命令を使用して PC(プログラムカウンタ)、SR(ステータスレジスタ)を回復し、割り込み関数から復帰してください。
- (c) C++プログラムで指定可能な関数メンバは、静的関数メンバです。例では、クラス A の静的関数メンバ g を割り込み関数として指定しています。非静的関数メンバは、指定できないので注意してください。

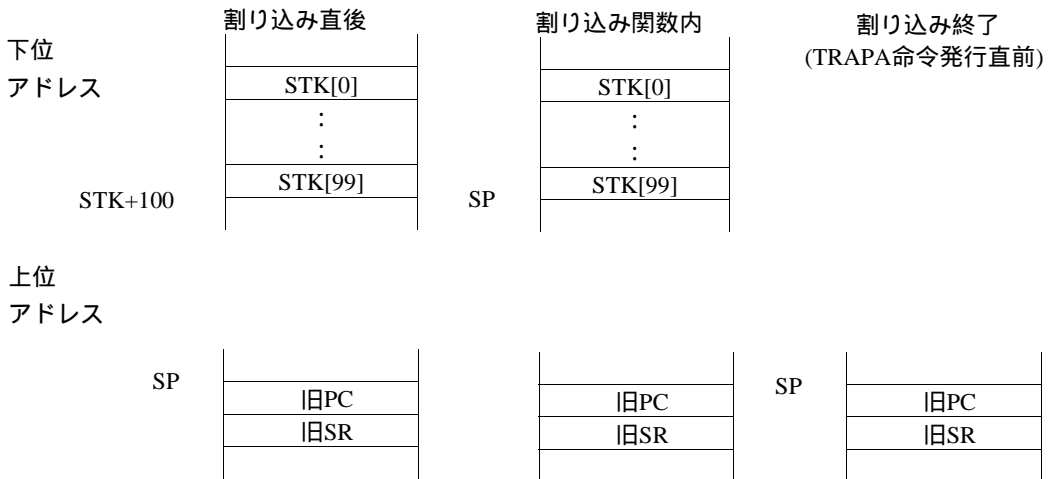


図 10.2 割り込み関数によるスタック使用例

## 備考

SH-3、SH-4 では割り込み時の動作が SH-1、SH-2、SH-2E の場合と異なるので、割り込みハンドラが必要になります。また、SH-3、SH-4 で `#pragma interrupt` と `#pragma noregsave` に同じ関数を指定した場合、その関数内で使用したレジスタのみ回避・回復を行います。

割り込み関数の定義に対して指定できる関数は、グローバル関数(C/C++プログラム)と静的関数メンバ(C++プログラム)です。グローバル関数は、`static` と指定しても `extern` として処理します。

また、関数の返すデータ型は `void` のみです。return 文のリターン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

```
#pragma interrupt (f1(sp=100), f2)
void f1(){...}.....(a)
int f2(){...}.....(b)
```

## 説明

(a) は正しい宣言になります。

(b) は関数の返すデータ型が `void` ではないので誤った宣言です。エラーメッセージを出力します。

割り込み関数として宣言した関数をプログラムの中で呼び出すことはできません。呼び出しがあった場合はエラーメッセージを出力します。ただし、割り込み関数として定義した関数を割り込み関数の宣言のないプログラム内で呼び出した場合は、エラーメッセージは出力しません。この場合、実行時の動作は保証しません。

例 1

割り込み関数宣言のある場合

```
#pragma interrupt (f1)
void f1(){...}
int f2(){f1();}.....(a)
```

説明

関数「f1」は割り込み関数として宣言しているのでプログラム中で呼び出すことはできません。(a)に対してエラーメッセージを出力します。

例 2

割り込み関数宣言がない場合

```
int f1();
int f2(){f1();}.....(b)
```

説明

関数「f1」は割り込み関数としての宣言がないので非割り込み関数 `int f1();` としてオブジェクトを生成します。関数「f1」が別コンパイル単位で割り込み関数として宣言された場合、実行時の動作は保証しません。



## 関数インライン展開

**#pragma inline**

書 式     #`pragma inline (<関数名>[,...])`

説 明     #`pragma inline` を用いて、インライン展開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)` を指定した関数は、その関数を呼び出したところにインライン展開されます。

例       ソースプログラム

```
#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;
}
```

備 考     以下の場合にはインライン展開しません。

- #`pragma inline` 指定より前に関数の定義がある。
- 可変パラメタを持つ関数である。
- 関数内でパラメタのアドレスを参照している。
- 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、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も外部定義を生成します。各プログラムファイル中にインライン関数の実体の記述がある場合は、必ず関数の宣言に `static` を指定してください。 `static` を指定した場合は、外部定義を生成しません。 `inline(C++言語)` 指定された関数は、外部定義を生成しません。

**#pragma inline\_asm**

書 式 #pragma inline\_asm (<関数名>[(size=数値)] [,...])

説 明 #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。関数名には、グローバル関数のみ指定できます。関数メンバは指定できません。アセンブラ埋め込みインライン関数のパラメタは、通常の間数呼び出しと同様にレジスタ、あるいはスタックに設定されますので、inline\_asm 関数から参照することができます。アセンブラ埋め込みインライン関数のリターン値は R0、SH-2E、SH-4 のとき単精度浮動小数点型のリターン値は FR0、SH-4 のとき倍精度浮動小数点型のリターン値は DR0 に設定してください。オプションとの組み合わせによりリターン値を設定するレジスタは異なります。詳細は表 9.7 を参照してください。  
(size=数値) 指定で、アセンブラ埋め込みインライン関数のサイズが指定できます。

## 例 ソースプログラム

```
#pragma inline_asm(rotl)
static int rotl(int a)
{
 ROTL R4
 MOV R4,R0
}
int x;
main()
{
 x=0x55555555;
 x=rotl(x);
}
```

## 出力結果(一部)

```

:
_main ;function main
 ;frame size = 4
MOV.L R14,@-R15
MOV.L L220+2,R14 ;x
MOV.L L220+6,R3 ;H'55555555
MOV.L R3,@R14
MOV R3,R4
BRA L219
NOP
L220:
 .RES.W 1
 .DATA.L x
 .DATA.L H'55555555
L219:
 ROTL R4
 MOV R4,R0
 .ALIGN 4
 MOV.L R0,@R14
 RTS
 MOV.L @R15+,R14
 .SECTION B,DATA,ALIGN=4
_x: ;static: x
 .RES.L 1
 .END
```

備考 #pragma inline\_asm は、関数本体の定義の前に指定して下さい。  
#pragma inline\_asm で指定した関数に対しても外部定義を生成します。  
アセンブラ埋め込みインライン関数内でラベルを使用する場合、必ずローカルラベルを使用して下さい。  
アセンブラ埋め込みインライン関数内で R8 から R14 のレジスタを使用する場合は、アセンブラ埋め込みインライン関数の先頭と最後でこれらのレジスタの退避・回復が必要です。また、FR12 から FR15 (CPU が SH-2E、SH-4 の場合)、DR12 から DR14 (CPU が SH-4 の場合) のレジスタを使用する場合も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。  
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。  
本機能を使用する際は、code=asmcode を指定してコンパイルしてください。  
(size=数値) で指定する数値は、実際のオブジェクトサイズ以上の値を指定してください。オブジェクトサイズより小さい値を指定した場合、動作は保証しません。また、数値が浮動小数点または 0 以下の数値の場合、エラーとなります。  
#pragma global\_register 機能で指定したレジスタを本関数内で使用する場合も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退避・回復が必要です。  
指定可能な関数は、グローバル関数のみです。関数メンバは指定できません。  
リテラルプールを生成するような記述は使用しないでください(MOV.L #100000,R0 等)。

```
#pragma regsave
#pragma noregsave
#pragma noregalloc
```

書式 `#pragma regsave (<関数名>[,...])`  
`#pragma noregsave (<関数名>[,...])`  
`#pragma noregalloc (<関数名>[,...])`

説明 関数名には、グローバル関数及び関数メンバを指定できます。  
`#pragma regsave` で指定された関数は、関数の出入口で保証するレジスタ(表 9.5 参照)の退避・回復を行います。また関数呼び出しを越えて R8-R14(CPU が SH-2E、SH-4 の場合はさらに FR12-FR15)を割り付けないオブジェクトを生成します。  
`#pragma noregsave` で指定された関数は、関数の出入口で保証するレジスタの退避・回復を行いません。  
`#pragma noregalloc` で指定された関数は、関数の出入口で保証するレジスタの退避・回復を行いません。また、関数呼び出しを越えて R8-R14(CPU が SH-2E、SH-4 の場合はさらに FR12-FR15)を割り付けないオブジェクトを生成します。  
`#pragma regsave` と `#pragma noregalloc` は同一関数に対して重複指定できます。このとき、関数の出入口で保証するレジスタ R8-R14(CPU が SH-2E、SH-4 の場合はさらに FR12-FR15)を全て退避・回復し、関数呼び出しがあるとき、レジスタ R8-R14(CPU が SH-2E、SH-4 の場合はさらに FR12-FR15)を割り付けないオブジェクトを生成します。  
`#pragma noregsave` が指定された関数は、下記の条件で使用することができます。

- 他の関数から呼び出されることなく、最初に起動する関数として使用する。
- `#pragma regsave` を指定した関数から呼び出す。
- `#pragma regsave` を指定した関数から、さらに `#pragma noregalloc` を介して呼び出す。

例

```
#pragma noregsave(f,A::j)
#pragma noregalloc(g)
#pragma regsave(h)
class A{
public:
 static void j();
};
void f();
void g();
void h();
void h() {
 g();
 f(); /* #pragma regsave 関数(h)から#pragma noregsave 関数(f)の直後 */
 /* の呼び出し */
}
void g() {
 f(); /* #pragma regsave 関数(h)から#pragma noregsave 関数(f,A::j) */
 /* の#pragma noregalloc 関数(g)を介した呼び出し */
 A::j();
}

void f(){}
```

備考 上記以外の方法で `#pragma noregsave` を指定した関数を呼び出した場合の結果は保証されませんので注意が必要です。

## エン트리関数作成

**#pragma entry**

書 式 `#pragma entry <関数名>[(sp=<定数>)]`

説 明 <関数名>で指定した関数をエン트리関数として扱います。  
 エン트리関数では、レジスタの回避・回復コードを一切作成しません。  
 CPU が SH-3、SH-4 の場合では、`sp=<定数>` 指定、もしくは `#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。

例 1

```
#pragma entry INIT(sp=0x10000)
void INIT() {
 :
}
```

<コード展開例>

```
.SECTION P, CODE
__INIT:
 MOV.L L1, R15
 :
L1: .DATA.L H'00010000
 :
```

例 2

```
#pragma stacksize 100
#pragma entry INIT
void INIT() {
 :
}
```

<コード展開例>

```
.SECTION S, STACK
.RES.B 100
.SECTION P, CODE
__INIT:
 MOV.L L1, R15
 :
L1: .DATA.L STARTOF S + SIZEOF S
 :
```

備 考 `#pragma entry` 指定は、関数の宣言前に行ってください。  
 エン트리関数はロードモジュール全体で 2 つまで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。  
`cpu=sh1|sh2|sh2e` を指定した場合、`sp=<定数>` 指定は無効になります。

## (3) その他の拡張機能

## グローバル変数レジスタ割り付け

**#pragma global\_register**

書 式 #pragma global\_register (<変数名>=<レジスタ名>[,...])

説 明 <変数名>で指定した変数に、<レジスタ名>で指定したレジスタを割り付けます。  
変数名には、グローバル変数及び静的データメンバを指定できます。

例

```
#pragma global_register(a=R8,A::b=R9)
class A{
public:
static int b;
};
int a;
void g()
{
a=A::b;
}
```

備 考 グローバル変数で、単純型またはポインタ型の変数に使用できます。また、double=floatを指定した場合を除き、double型の変数は指定できません(CPUがSH-4の場合を除く)。指定可能なレジスタは、R8-R14、FR12-FR15(CPUがSH-2E、SH-4の場合)、DR12-DR14(CPUがSH-4の場合)です。

初期値の設定はできません。また、アドレスの参照もできません。

指定された変数の、リンク先からの参照は保証されません。

静的データメンバの指定は可能ですが、非静的データメンバの指定は不可能です。

FR12-FR15に設定可能な変数の型

SH-2Eの場合

- ・float型変数
- ・double型変数(double=float指定)

SH-4の場合

- ・float型変数(fpu=double指定なし)
- ・double型変数(fpu=single指定)

DR12~DR14に設定可能な変数の型

SH-4の場合

- ・float型変数(fpu=double指定)
- ・double型変数(fpu=single指定なし)

## GBR ベース変数指定

**#pragma gbr\_base**  
**#pragma gbr\_base1**

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 書 式 | <code>#pragma gbr_base(変数名[,...])</code><br><code>#pragma gbr_base1(変数名[,...])</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| 説 明 | <p>変数を GBR レジスタからのオフセットでアクセスすることを指定します。<br/>変数名に、変数及び静的データメンバを指定できます。<br/>#pragma gbr_base で指定した変数は、セクション\$G0 に割り付けられます。<br/>#pragma gbr_base1 で指定した変数は、セクション\$G1 に割り付けられます。<br/>#pragma gbr_base は、変数が GBR レジスタの指すアドレスからオフセット 0-127 バイトにあることを指定します。<br/>#pragma gbr_base1 は、#pragma gbr_base でアクセス可能でない範囲 (GBR レジスタの指すアドレスからオフセット 128 バイト以上) の変数に対して指定できます。GBR レジスタの指すアドレスからのオフセットが、char 型、unsigned char 型の場合は最大 255 バイト、short 型、unsigned short 型の場合は、最大 510 バイト、int 型、unsigned int 型、long 型、unsigned long 型、float 型、double 型の場合は最大 1020 バイトであることを指定します。<br/>コンパイラは、これらの指定に基づき、変数の参照、設定に対して、最適な GBR 相対アドレッシングでオブジェクトプログラムを生成します。また、セクション\$G0 内の char 型、unsigned char 型のデータに対して、GBR 間接アドレッシングで最適なビット命令を生成します。</p> |
| 備 考 | <p>セクション\$G0 のリンク後のサイズ合計が 128 バイトを超えた場合は動作を保証しません。また、セクション\$G1 内に、上記の #pragma gbr_base1 の制約で各データ型に示した以上のオフセットを持つデータがある場合、動作を保証しません。<br/>セクション\$G1 は、リンク時にセクション\$G0 の 128 バイト後に必ず配置してください。本機能を使用する場合は、プログラム実行開始時に、GBR レジスタにセクション\$G0 の先頭を設定してください。<br/>静的データメンバは指定可能ですが、非静的データメンバは指定できません。<br/>gbr=auto を指定した場合、#pragma gbr_base、#pragma gbr_base1 指定は無効になります。</p>                                                                                                                                                                                                                                                                                                                                                 |

## 10.2.2 セクションアドレス演算子

## セクションアドレス演算子

---

***\_\_sectop***  
***\_\_secend***

---

書 式    `__sectop("<セクション名>")`  
          `__secend("<セクション名>")`

説 明    `__sectop` で指定した<セクション名>の先頭アドレスを参照します。  
          `__secend` で指定した<セクション名>の末尾+1 アドレスを参照します。

例

```
#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 0x100
#pragma entry INIT
void main(void);
void INIT(void)
{
 _INITSCT();
 main();
 sleep();
}
```



### 10.2.3 組み込み関数

C/C++言語で記述できない以下の機能を、組み込み関数として提供します。

- ステータスレジスタの設定、参照
- ベクタベースレジスタの設定、参照
- グローバルベースレジスタを利用した I/O 機能
- C/C++言語で使用するレジスタ資源と競合しないシステム命令
- 浮動小数点ユニットを利用したマルチメディア命令、コントロールレジスタの設定、参照

組み込み関数は、通常の関数と同様に関数呼び出し形式で記述します。

組み込み関数の一覧を表 10.25 に示します。

表 10.25 組み込み関数の一覧

| 項目 | 仕様                                                                                        | 機能               |
|----|-------------------------------------------------------------------------------------------|------------------|
| 1  | void set_cr(int cr)                                                                       | SR の設定           |
| 2  | int get_cr(void)                                                                          | SR の参照           |
| 3  | void set_imask(int mask)                                                                  | 割り込みマスクの設定       |
| 4  | int get_imask(void)                                                                       | 割り込みマスクの参照       |
| 5  | void set_vbr(void *base)                                                                  | VBR の設定          |
| 6  | void *get_vbr(void)                                                                       | VBR の参照          |
| 7  | void set_gbr(void *base)                                                                  | GBR の設定          |
| 8  | void *get_gbr(void)                                                                       | GBR の参照          |
| 9  | unsigned char gbr_read_byte(int offset)                                                   | GBR ベースのバイト参照    |
| 10 | unsigned short gbr_read_word(int offset)                                                  | GBR ベースのワード参照    |
| 11 | unsigned long gbr_read_long(int offset)                                                   | GBR ベースのロングワード参照 |
| 12 | void gbr_write_byte(int offset, unsigned char data)                                       | GBR ベースのバイト設定    |
| 13 | void gbr_write_word(int offset, unsigned short data)                                      | GBR ベースのワード設定    |
| 14 | void gbr_write_long(int offset, unsigned long data)                                       | GBR ベースのロングワード設定 |
| 15 | void gbr_and_byte(int offset, unsigned char mask)                                         | GBR ベースのバイト AND  |
| 16 | void gbr_or_byte(int offset, unsigned char mask)                                          | GBR ベースのバイト OR   |
| 17 | void gbr_xor_byte(int offset, unsigned char mask)                                         | GBR ベースのバイト XOR  |
| 18 | int gbr_tst_byte(int offset, unsigned char mask)                                          | GBR ベースのバイト TEST |
| 19 | void sleep (void)                                                                         | SLEEP 命令         |
| 20 | int tas (char *addr)                                                                      | TAS 命令           |
| 21 | int trapa (int trap_no)                                                                   | TRAPA 命令         |
| 22 | int trapa_svc (int trap_no, int code, type1 para1, type2 para2, type3 para3, type4 para4) | OS システムコール       |
| 23 | void prefetch (void *p)                                                                   | PREF 命令          |
| 24 | void trace(long v)                                                                        | TRACE 命令         |
| 25 | void ldtlb(void)                                                                          | LDTLB 命令         |
| 26 | void nop(void)                                                                            | NOP 命令           |

## 10. C/C++言語仕様

| 項目 | 仕様                                                                                                                                              | 機能                                 |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------|
| 27 | long dmuls_h(long data1, long data2)                                                                                                            | 符号付き 64bit 乗算の上位 32bit             |
| 28 | unsigned long dmuls_l(long data1, long data2)                                                                                                   | 符号付き 64bit 乗算の下位 32bit             |
| 29 | 64bit 乗算<br>unsigned long dmulu_h(unsigned long data1,<br>unsigned long data2)                                                                  | 符号なし 64bit 乗算の上位 32bit             |
| 30 | unsigned long dmulu_l(unsigned long data1,<br>unsigned long data2)                                                                              | 符号なし 64bit 乗算の下位 32bit             |
| 31 | unsigned short swapb(unsigned short data)                                                                                                       | SWAP.B 命令                          |
| 32 | データ上位・下位<br>交換<br>unsigned long swapw(unsigned long data)                                                                                       | SWAP.W 命令                          |
| 33 | unsigned long end_cnv(unsigned long data)                                                                                                       | 4 バイトデータの上位・下位交換                   |
| 34 | 積和演算<br>int macw(short *ptr1, short *ptr2, unsigned int count)<br>int macwl(short *ptr1, short *ptr2,<br>unsigned int count, unsigned int mask) | MAC.W 命令                           |
| 35 | int macll(int *ptr1, int *ptr2, unsigned int count,<br>unsigned int mask)                                                                       | MAC.L 命令                           |
| 36 | 浮動小数点ユニ<br>ット<br>void set_fpscr(int cr)                                                                                                         | FPSCR の設定                          |
| 37 | int get_fpscr(void)                                                                                                                             | FPSCR の参照                          |
| 38 | float fipr(float vect1[4], float vect2[4])                                                                                                      | FIPR 命令                            |
| 39 | void ftrv(float vec1[4], float vec2[4])                                                                                                         | FTRV 命令                            |
| 40 | void ftrvadd(float vec1[4], float vec2[4],<br>float vec3[4])                                                                                    | 4 次元ベクタの 4 × 4 行列による変換と 4 次元ベクタとの和 |
| 41 | void ftrvsub(float vec1[4], float vec2[4],<br>float vec3[4])                                                                                    | 4 次元ベクタの 4 × 4 行列による変換と 4 次元ベクタとの差 |
| 42 | 単精度浮動小数<br>点ベクタ演算<br>void add4(float vec1[4], float vec2[4],<br>float vec3[4])                                                                  | 4 次元ベクタの和                          |
| 43 | void sub4(float vec1[4], float vec2[4],<br>float vec3[4])                                                                                       | 4 次元ベクタの差                          |
| 44 | void mtrx4mul(float mat1[4][4], float mat2[4][4])                                                                                               | 4 × 4 行列の乗算                        |
| 45 | void mtrx4muladd(float mat1[4][4], float mat2[4][4],<br>float mat3[4][4])                                                                       | 4 × 4 行列の乗算と和                      |
| 46 | void mtrx4mulsub(float mat1[4][4], float mat2[4][4],<br>float mat3[4][4])                                                                       | 4 × 4 行列の乗算と差                      |
| 47 | 拡張レジスタへ<br>のアクセス<br>void ld_ext(float mat[4][4])                                                                                                | 拡張レジスタへのロード                        |
| 48 | void st_ext(float mat[4][4])                                                                                                                    | 拡張レジスタからのストア                       |

組み込み関数を使用する場合は、必ず<machine.h>、または<umachine.h>や<smachine.h>をインクルードしてください。

SH-3、SH-4 の実行モードに対応し<machine.h>の内容を表 10.26 のように分割しています。

表 10.26 組み込み関数用インクルードファイル

|   | インクルードファイル   | 内容                  |
|---|--------------|---------------------|
| 1 | <machine.h>  | 組み込み関数全体            |
| 2 | <smachine.h> | 特権モードでのみ使用可能な組み込み関数 |
| 3 | <umachine.h> | 2 以外の組み込み関数         |

---

***void set\_cr(int cr)***

---

説明 ステータスレジスタに `cr` (32 ビット) を設定します。

ヘッダ <machine.h>または<smachine.h>

引数 `cr` 設定値

例

```
#include <machine.h>
void main(void)
{
 set_cr(0x60000000); /* Supervisor, RBank=1, BL=0, Imask=0 */
}
```

---

***int get\_cr(void)***

---

説明 ステータスレジスタの値を参照します。

ヘッダ <machine.h>または<smachine.h>

リターン値 ステータスレジスタの値

例

```
#include <machine.h>
void main(void)
{
 set_cr(get_cr()|0x1000000); /* set BL bit */
}
```

---

***void set\_imask(int mask)***

---

説明 割り込みマスク(4ビット)にmask(4ビット)を設定します。

ヘッダ <machine.h>または<smachine.h>

引数 mask 設定値(4ビット)

例

```
#include <machine.h>
void main(void)
{
 set_imask(15);
}
```

---

***int get\_imask(void)***

---

説明 割り込みマスク(4ビット)を参照します。

ヘッダ <machine.h>または<smachine.h>

リターン値 割り込みマスクの値

例

```
#include <machine.h>
void main(void)
{
 int mask;
 mask=get_imask();
}
```

**VBR 設定*****void set\_vbr(void \*base)***

説明      ベクタベースレジスタ (VBR) に base (32 ビット) を設定します。

ヘッダ     <machine.h>または<smachine.h>

引 数      base                      設定値

例

```
#include <machine.h>
#define VBR 0x0000FC00
void main(void)
{
 set_vbr((void *)VBR);
}
```

**VBR 参照*****void \*get\_vbr(void)***

説明      ベクタベースレジスタ (VBR) を参照します。

ヘッダ     <machine.h>または<smachine.h>

リターン値      ベクタベースレジスタの値

例

```
#include <machine.h>
void main(void)
{
 void *vbr;
 vbr=get_vbr();
}
```

---

**void set\_gbr(void \*base)**

---

説明 グローバルベースレジスタ (GBR) に base (32 ビット) を設定します。

ヘッダ <machine.h>または<umachine.h>

引数 base 設定値

例

```
#include <machine.h>
#define IOBASE 0x05fffec0
void main(void)
{
 set_gbr((void *)IOBASE);
}
```

備考 GBR はコントロールレジスタのため、本コンパイラでは関数ごとに内容を保証していません。GBR の設定を変えるときには注意が必要です。  
gbr=auto を指定した場合、本関数は使用できません。

---

**void \*get\_gbr(void)**

---

説明 グローバルベースレジスタ (GBR) の値を参照します。

ヘッダ <machine.h>または<umachine.h>

リターン値 グローバルベースレジスタの値

例

```
#include <machine.h>
void main(void)
{
 void *gbr;
 gbr=get_gbr();
}
```

備考 gbr=auto を指定した場合、本関数は使用できません。

## GBR ベースバイト参照

***unsigned char gbr\_read\_byte(int offset)***

|       |                                                                                                                       |           |
|-------|-----------------------------------------------------------------------------------------------------------------------|-----------|
| 説明    | GBR 相対 offset のバイトデータ (8 ビット) を参照します。                                                                                 |           |
| ヘッダ   | <machine.h>または<umachine.h>                                                                                            |           |
| リターン値 | バイトデータ (8 ビット) の参照値                                                                                                   |           |
| 引数    | offset                                                                                                                | オフセットアドレス |
| 例     | <pre>#include &lt;machine.h&gt; #define BDATA 0 void main(void) {     if(gbr_read_byte(BDATA) != 0)         : }</pre> |           |
| 備考    | offset は定数でなければなりません。<br>offset 指定可能範囲は+255 バイトまでです。<br>gbr=auto を指定した場合、本関数は使用できません。                                 |           |

## GBR ベースワード参照

***unsigned short gbr\_read\_word(int offset)***

|       |                                                                                                                       |           |
|-------|-----------------------------------------------------------------------------------------------------------------------|-----------|
| 説明    | GBR 相対 offset のワードデータ (16 ビット) を参照します。                                                                                |           |
| ヘッダ   | <machine.h>または<umachine.h>                                                                                            |           |
| リターン値 | バイトデータ (16 ビット) の参照値                                                                                                  |           |
| 引数    | offset                                                                                                                | オフセットアドレス |
| 例     | <pre>#include &lt;machine.h&gt; #define WDATA 0 void main(void) {     if(gbr_read_word(WDATA) != 0)         : }</pre> |           |
| 備考    | offset は定数でなければなりません。<br>offset 指定可能範囲は+510 バイトまでです。<br>gbr=auto を指定した場合、本関数は使用できません。                                 |           |

***unsigned long gbr\_read\_long(int offset)***

|       |                                                                                                                     |           |
|-------|---------------------------------------------------------------------------------------------------------------------|-----------|
| 説明    | GBR 相対 offset のロングワードデータ (32 ビット) を参照します。                                                                           |           |
| ヘッダ   | <machine.h>または<umachine.h>                                                                                          |           |
| リターン値 | バイトデータ (32 ビット) の参照値                                                                                                |           |
| 引数    | offset                                                                                                              | オフセットアドレス |
| 例     | <pre>#include &lt;machine.h&gt; #define LDATA 0 void main(void) {     if(gbr_read_long(LDATA)!=0)         : }</pre> |           |
| 備考    | offset は定数でなければなりません。<br>offset 指定可能範囲は+1020 バイトまでです。<br>gbr=auto を指定した場合、本関数は使用できません。                              |           |

***void gbr\_write\_byte(int offset, unsigned char data)***

|     |                                                                                                        |             |
|-----|--------------------------------------------------------------------------------------------------------|-------------|
| 説明  | GBR 相対 offset へ data (8 ビット) を設定します。                                                                   |             |
| ヘッダ | <machine.h>または<umachine.h>                                                                             |             |
| 引数  | offset                                                                                                 | オフセットアドレス   |
|     | data                                                                                                   | 設定値 (8 ビット) |
| 例   | <pre>#include &lt;machine.h&gt; #define BDATA 0 void main(void) {     gbr_write_byte(BDATA,0); }</pre> |             |
| 備考  | offset は定数でなければなりません。<br>offset 指定可能範囲は+255 バイトまでです。<br>gbr=auto を指定した場合、本関数は使用できません。                  |             |



## GBR ベースワード設定

***void gbr\_write\_word(int offset, unsigned short data)***

説明 GBR 相対 offset へ data(16 ビット)を設定します。

ヘッダ <machine.h>または<umachine.h>

引数      offset                      オフセットアドレス  
          data                      設定値(16 ビット)

```
例
#include <machine.h>
#define WDATA 0
void main(void)
{
 gbr_write_word(WDATA,0);
}
```

備考 offset は定数でなければなりません。  
offset 指定可能範囲は+510 バイトまでです。  
gbr=auto を指定した場合、本関数は使用できません。

## GBR ベースロングワード設定

***void gbr\_write\_long(int offset, unsigned long data)***

説明 GBR 相対 offset へ data(32 ビット)を設定します。

ヘッダ <machine.h>または<umachine.h>

引数      offset                      オフセットアドレス  
          data                      設定値(32 ビット)

```
例
#include <machine.h>
#define LDATA 0
void main(void)
{
 gbr_write_long(LDATA,0);
}
```

備考 offset は定数でなければなりません。  
offset 指定可能範囲は+1020 バイトまでです。  
gbr=auto を指定した場合、本関数は使用できません。

---

***void gbr\_and\_byte(int offset, unsigned char mask)***

---

説明 GBR 相対 offset のバイトデータと mask の AND をとり、offset に設定します。

ヘッダ <machine.h>または<umachine.h>

引数 offset                    オフセットアドレス  
mask                         データ(8ビット)

例

```
#include <machine.h>
#define BDATA 0
void main(void)
{
 gbr_and_byte(BDATA, 0x01);
}
```

備考 mask は定数でなければなりません。  
offset 指定可能範囲は+255 バイトまでです。  
mask 指定可能範囲は 0 ~ +255 です。  
gbr=auto を指定した場合、本関数は使用できません。

---

***void gbr\_or\_byte(int offset, unsigned char mask)***

---

説明 GBR 相対 offset のバイトデータと mask の OR をとり、offset に設定します。

ヘッダ <machine.h>または<umachine.h>

引数 offset                    オフセットアドレス  
mask                         データ(8ビット)

例

```
#include <machine.h>
#define BDATA 0
void main(void)
{
 gbr_or_byte(BDATA, 0x01);
}
```

備考 mask は定数でなければなりません。  
offset 指定可能範囲は+255 バイトまでです。  
mask 指定可能範囲は 0 ~ +255 です。  
gbr=auto を指定した場合、本関数は使用できません。

**GBR ベースバイト XOR*****void gbr\_xor\_byte(int offset, unsigned char mask)***

説明 GBR 相対 offset のバイトデータと mask の XOR をとり、offset に設定します。

ヘッダ <machine.h>または<umachine.h>

引数 offset オフセットアドレス  
mask データ(8ビット)

```
例
#include <machine.h>
#define BDATA 0
void main(void)
{
 gbr_xor_byte(BDATA, 0x01);
}
```

備考 mask は定数でなければなりません。  
offset 指定可能範囲は+255バイトまでです。  
mask 指定可能範囲は0~+255です。  
gbr=auto を指定した場合、本関数は使用できません。

**GBR ベースバイト TEST*****int gbr\_tst\_byte(int offset, unsigned char mask)***

説明 GBR 相対 offset のバイトデータと mask の AND をとり、その値を 0 と判定し結果を T ビットに設定します。

ヘッダ <machine.h>または<umachine.h>

引数 offset オフセットアドレス  
mask データ(8ビット)

```
例
#include <machine.h>
#define BDATA 0
int a;
void main(void)
{
 if(gbr_tst_byte(BDATA, 0))
 a = 0;
}
```

備考 mask は定数でなければなりません。  
offset 指定可能範囲は+255バイトまでです。  
mask 指定可能範囲は0~+255です。  
gbr=auto を指定した場合、本関数は使用できません。

## GBR 組み込み関数の使用例

```
#include <machine.h>
#define CDATA1 0
#define CDATA2 1
#define CDATA3 2
#define SDATA1 4
#define IDATA1 8
#define IDATA2 12

struct {
 char cdata1; /*offset 0 */
 char cdata2; /*offset 1 */
 char cdata3; /*offset 2 */
 short sdata1; /*offset 4 */
 int idata1; /*offset 8 */
 int idata2; /*offset 12 */
} table;
void f();

void f()
{
 set_gbr(&table); /* GBR に table の先頭アドレス */
 : /* を設定 */
 gbr_write_byte(CDATA2,10); /* table.cdata2 に 10 を設定 */
 gbr_write_long(IDATA2,100); /* table.idata2 に 100 を設定 */
 :
 if(gbr_read_byte(CDATA2)!=10) /* table.cdata2 の値を参照 */
 gbr_and_byte(CDATA2,10); /* table.cdata2 の値と 10 の AND */
 : /* をとって table.cdata2 に設定 */
 gbr_or_byte(CDATA2,0x0F); /* table.cdata2 の値と 0x0F の OR */
 : /* をとって table.cdata2 に設定 */
 sleep(); /* SLEEP 命令に展開 */
}
```

## GBR 組み込み関数の有効な使用法

- 頻繁にアクセスするオブジェクトをメモリに割り付け、そのオブジェクトの先頭アドレスを GBR に設定する。
- 論理演算を多用するバイトデータをできるだけ構造体の先頭から 128 バイトまでに宣言する。

これにより、構造体アクセスに必要な先頭アドレスロードと、論理演算に必要なメモリロード、ストアに対する命令が削減できます。

**SLEEP 命令*****void sleep(void)***

説明 低消費電力状態遷移命令 SLEEP に展開します。

ヘッダ <machine.h>または<smachine.h>

例

```
#include <machine.h>
void main(void)
{
 sleep();
}
```

**TAS 命令*****int tas(char \*addr)***

説明 TAS.B @Rn に展開します。

ヘッダ <machine.h>または<umachine.h>

引数 addr TAS 命令で指定するアドレス

例

```
#include <machine.h>
char a;
void main(void)
{
 tas(&a);
}
```

***int trapa (int trap\_no)***

説明 TRAPA #trap\_no に展開します。

ヘッダ <machine.h>または<umachine.h>

引数 trap\_no                    トラップ番号

例

```
#include <machine.h>
void main(void)
{
 trapa(0);
}
```

***int trapa\_svc (int trap\_no, int code, type1 para1, type2 para2, type3 para3, type4 para4)***

説明 HI7000 をはじめ、各種 OS のシステムコールを発行します。trapa\_svc を実行すると、R0 に code、R4~R7 に para1~para4 を設定し、TRAPA #trap\_no 命令を実行します。

ヘッダ <machine.h>または<umachine.h>

引数 trap\_no                    トラップ番号  
code                            機能コード  
para1~para4                    パラメタ(0~4個の可変)  
                                  型 type1~type4 は、汎整数型またはポインタ型です。

例

```
#include <machine.h>
#define SIG_SEM 0xffc8
void main(void)
{
 trapa_svc(63, SIG_SEM, 0x05);
}
```

**PREF 命令*****void prefetch (void \*p)***

説明 p の指す領域(16 バイト、ただし、領域は `(int)p&0xffffffff0`)からの 16 バイトのデータをキャッシュに読み込みます。

ヘッダ <machine.h>または<umachine.h>

引数 p                               プリフェッチを行うアドレス

例

```
#include <machine.h>
char a[1200];
void main(void)
{
 int *pa = a;
 prefetch(pa);
}
```

備考 `cpu=sh3|sh4` を指定した場合のみ使用可能です。  
プログラムの論理的な動作には影響を与えません。

**TRACE 命令*****void trace (long v)***

説明 TRACE 命令に展開します。

ヘッダ <machine.h>または<umachine.h>

引数 v                               出力する変数

例

```
#include <machine.h>
void main(void)
{
 long v;
 trace(v);
}
```

備考 `cpu=sh3|sh4` を指定した場合のみ使用可能です。

***void ldtlb (void)***

---

- 説 明 LDTLB 命令に展開します。
- ヘッダ <machine.h>または<smachine.h>
- 例
- ```
#include <machine.h>
void main(void)
{
    ldtlb();
}
```
- 備 考 cpu=sh3|sh4 を指定した場合のみ使用可能です。

void nop (void)

- 説 明 NOP 命令に展開します。
- ヘッダ <machine.h>または<umachine.h>
- 例
- ```
#include <machine.h>
void main(void)
{
 int a;
 if (a) {
 nop();
 }
}
```



---

***long dmuls\_h (long data1, long data2)***

---

説明 符号付き 32 ビット×符号付き 32 ビット 符号付き 64 ビットの乗算を行い、結果の上位 32 ビットを参照します。

ヘッダ <machine.h>または<umachine.h>

例

```
#include <machine.h>
extern long data1, data2;
extern long result;
void main(void)
{
 result = dmuls_h(data1, data2);
}
```

備考 cpu=sh1 を指定した場合、本関数は使用できません。

---

***unsigned long dmuls\_l (long data1, long data2)***

---

説明 符号付き 32 ビット×符号付き 32 ビット 符号付き 64 ビットの乗算を行い、結果の下位 32 ビットを参照します。

ヘッダ <machine.h>または<umachine.h>

例

```
#include <machine.h>
extern long data1, data2;
extern unsigned long result;
void main(void)
{
 result = dmuls_l(data1, data2);
}
```

備考 cpu=sh1 を指定した場合、本関数は使用できません。

***unsigned long dmulu\_h (unsigned long data1, unsigned long data2)***

説明 符号なし 32 ビット×符号なし 32 ビット 符号なし 64 ビットの乗算を行い、結果の上位 32 ビットを参照します。

ヘッダ <machine.h>または<umachine.h>

```
例
#include <machine.h>
extern unsigned long data1, data2;
extern unsigned long result;
void main(void)
{
 result = dmulu_h(data1, data2);
}
```

備考 cpu=sh1 を指定した場合、本関数は使用できません。

***unsigned long dmulu\_l (unsigned long data1, unsigned long data2)***

説明 符号なし 32 ビット×符号なし 32 ビット 符号なし 64 ビットの乗算を行い、結果の下位 32 ビットを参照します。

ヘッダ <machine.h>または<umachine.h>

```
例
#include <machine.h>
extern unsigned long data1, data2;
extern unsigned long result;
void main(void)
{
 result = dmulu_l(data1, data2);
}
```

備考 cpu=sh1 を指定した場合、本関数は使用できません。

**SWAP.B 命令*****unsigned short swapb(unsigned short data)***

説明 2バイトデータの上位・下位1バイトを交換します。

ヘッダ <machine.h>または<umachine.h>

例

```
#include <machine.h>
extern unsigned short data;
extern unsigned short result;
void main(void)
{
 result = swapb(data);
 /* 例えば、data=0x1234の場合、
 result=0x3412になります。 */
}
```

**SWAP.W 命令*****unsigned long swapw(unsigned long data)***

説明 4バイトデータの上位・下位2バイトを交換します。

ヘッダ <machine.h>または<umachine.h>

例

```
#include <machine.h>
extern unsigned long data;
extern unsigned long result;
void main(void)
{
 result = swapw(data);
 /* 例えば、data=0x12345678の場合、
 result=0x56781234になります。 */
}
```

***unsigned long end\_cnv1(unsigned long data)***

説明 4 バイトデータを 1 バイトごとに上位・下位を逆順に並べ替えます。

ヘッダ <machine.h>または<umachine.h>

```
例
#include <machine.h>
extern unsigned long data;
extern unsigned long result;
void main(void)
{
 result = end_cnv1(data);
 /* 例えば、data=0x12345678 の場合、
 result=0x78563412 になります。 */
}
```

**MAC.W 命令*****int macw(short \*ptr1, short \*ptr2, unsigned int count)******int macwl(short \*ptr1, short \*ptr2, unsigned int count, unsigned int mask)***

説明 二つのデータテーブルの内容の積和を求めます。

ヘッダ <machine.h>または<umachine.h>

リターン値 演算結果

|    |       |                   |
|----|-------|-------------------|
| 引数 | ptr1  | 積和演算するデータの先頭アドレス  |
|    | ptr2  | 積和演算するデータの先頭アドレス  |
|    | count | 積和演算を実行する回数       |
|    | mask  | リングバッファ対応のアドレスマスク |

```
例
#include <machine.h>
short tbl1[]={a1,a2,a3,a4};
short tbl2[]={b1,b2,b3,b4};
int result1,result2;
void main(void)
{
 result1=macw(tbl1,tbl2,3); /* a1*b1+a2*b2+a3*b3 を求めます。 */

 result2=macwl(tbl1,tbl2,4,0xffffffff);
 /* a1*b1+a2*b2+a3*b1+a4*b2 を求めます。 */
}
```

備考 パラメタのチェックを行いませんので、次のことに注意してください。

- ・ ptr1、ptr2 の指すテーブルは、2 バイトで境界整合されていなければなりません。
- ・ macwl の ptr2 の指すテーブルはリングバッファマスク × 2 のサイズで境界整合されていなければなりません。

## MAC.L 命令

---

***int macl(int \*ptr1, int \*ptr2, unsigned int count)***  
***int macll(int \*ptr1, int \*ptr2, unsigned int count, unsigned int mask)***

---

説明 二つのデータテーブルの内容の積和を求めます。

ヘッダ <machine.h>または<umachine.h>

リターン値 演算結果

|    |       |                   |
|----|-------|-------------------|
| 引数 | ptr1  | 積和演算するデータの先頭アドレス  |
|    | ptr2  | 積和演算するデータの先頭アドレス  |
|    | count | 積和演算を実行する回数       |
|    | mask  | リングバッファ対応のアドレスマスク |

例

```
#include <machine.h>
int tbl1[]={a1,a2,a3,a4};
int tbl2[]={b1,b2,b3,b4};
int result1,result2;
void main(void)
{
 result1=macl(tbl1,tbl2,3); /* a1*b1+a2*b2+a3*b3 を求めます。*/
 result2=macll(tbl1,tbl2,4,0xffffffff);
 /* a1*b1+a2*b2+a3*b1+a4*b2 を求めます。*/
}
```

備考 cpu=sh1 を指定した場合、本関数は使用できません。  
 パラメタのチェックを行いませんので、次のことに注意してください。

- ・ ptr1、 ptr2 の指すテーブルは、4 バイトで境界整合されていなければなりません。
- ・ macll の ptr2 の指すテーブルはリングバッファマスク × 2 のサイズで境界整合されていなければなりません。

## FPSCR 設定

---

***void set\_fpscr(int cr)***

---

説明 浮動小数点ステータス制御レジスタ (FPSCR) に cr (32 ビット) を設定します。

ヘッダ <machine.h>または<umachine.h>

引数 cr 設定値 (32 ビット)

例

```
#include <machine.h>
void main(void)
{
 set_fpscr(0);
}
```

備考 cpu=sh2e|sh4 を指定した場合のみ使用可能です。

***int get\_fpscr(void)***

説明 浮動小数点ステータス制御レジスタ (FPSCR) を参照します。

ヘッダ <machine.h>または<umachine.h>

リターン値 FPSCR の値

```
例
#include <machine.h>
int cr;
void main(void)
{
 cr=get_fpscr();
}
```

備考 cpu=sh2e|sh4 を指定した場合のみ使用可能です。

***float fipr(float vect1[4], float vect2[4])***

説明 2つのベクタの内積を求めます。

ヘッダ <machine.h>または<umachine.h>

リターン値 演算結果

|    |       |     |
|----|-------|-----|
| 引数 | vect1 | ベクタ |
|    | vect2 | ベクタ |

```
例
#include <machine.h>
extern float data1[4],data2[4];
float result;
void main(void)
{
 result=fipr(data1,data2);
}
```

備考 cpu=sh4 を指定した場合のみ使用可能です。

**void ftrv(float vec1[4], float vec2[4])**

**説明** vec1(ベクタ)を tbl(4×4 行列)で変換した結果を vec2(ベクタ)に格納します。  
tbl は組み込み関数 ld\_ext()でロードした行列です。

**ヘッダ** <machine.h>または<umachine.h>

**引数** vec1                   ベクタ  
vec2                   ベクタ

**例**

```
#include <machine.h>
extern float tbl[4][4];
extern float data1[4],data2[4];
void main(void)
{
 ld_ext(tbl);
 ftrv(data1,data2);
 /* i=0,1,2,3 として
 data2[i]=data1[0]*tbl[0][i]+data1[1]*tbl[1][i]
 +data1[2]*tbl[2][i]+data1[3]*tbl[3][i]
 が data2 の結果になります。 */
}
```

**備考** cpu=sh4 を指定した場合のみ使用可能です。  
ld\_ext()関数および st\_ext()関数は、浮動小数点ステータス制御レジスタ(FPSCR)の浮動小数点レジスタバンクビット(FR)を変更して拡張レジスタにアクセスします。割り込み関数内で ld\_ext() 関数または st\_ext()関数を使用しているときは、ベクタ演算組み込み関数の前後で割り込みマスクを変更してください。以下に例を示します。

**例:**

```
extern float mat1[4][4];
extern float vec1[4],vec2[4];
#pragma interrupt (intfunc)
void intfunc(){
 :
 ld_ext();
 :
}
void normfunc(){
 :
 int maskdata=get_imask();
 set_imask(15);
 ld_ext(mat1);
 ftrv(vec1,vec2);
 set_imask(maskdata);
 :
}
```

## 4次元ベクタの4×4行列による変換と4次元ベクタとの和

***void ftrvadd(float vec1[4], float vec2[4], float vec3[4])***

説明 `vec1`(ベクタ)を `tbl`(4×4行列)で変換した結果と `vec2`(ベクタ)の和を `vec3`(ベクタ)に格納します。`tbl`は組み込み関数 `ld_ext()`でロードした行列です。

ヘッダ `<machine.h>`または`<umachine.h>`

引数 `vec1`                   ベクタ  
`vec2`                   ベクタ  
`vec3`                   ベクタ

例

```
#include <machine.h>
extern float tbl[4][4];
extern float data1[4];
extern float data2[4];
extern float data3[4];
void main(void)
{
 ld_ext(tbl);
 ftrvadd(data1,data2,data3); /* data3=data1×tbl+data2 */
 /* i=0,1,2,3として
 data3[i]=data1[0]*tbl[0][i]
 +data1[1]*tbl[1][i]
 +data1[2]*tbl[2][i]
 +data1[3]*tbl[3][i]
 +data2[i]
 が data3 の結果になります。 */
}
```

備考 `cpu=sh4`を指定した場合のみ使用可能です。



---

**4次元ベクタの4×4行列による変換と4次元ベクタとの差**

---

***void ftrvsub(float vec1[4], float vec2[4], float vec3[4])***

---

説明     vec1(ベクタ)をtbl(4×4行列)で変換した結果とvec2(ベクタ)の差をvec3(ベクタ)に格納します。tblは組み込み関数ld\_ext()でロードした行列です。

ヘッダ    <machine.h>または<umachine.h>

引 数     vec1                 ベクタ  
           vec2                 ベクタ  
           vec3                 ベクタ

例

```
#include <machine.h>
extern float tbl[4][4];
extern float data1[4];
extern float data2[4];
extern float data3[4];
void main(void)
{
 ld_ext(tbl);
 ftrvsub(data1,data2,data3); /* data3=data1×tbl - data2 */
 /* i=0,1,2,3として
 data3[i]=data1[0]*tbl[0][i]
 +data1[1]*tbl[1][i]
 +data1[2]*tbl[2][i]
 +data1[3]*tbl[3][i]
 -data2[i]
 がdata3の結果になります。 */
}
```

備 考     cpu=sh4を指定した場合のみ使用可能です。

## 4次元ベクタの和

***void add4(float vec1[4], float vec2[4], float vec3[4])***

---

|     |                                                                                                                                                                                          |     |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 説明  | vec1(ベクタ)とvec2(ベクタ)の和をvec3(ベクタ)に格納します。                                                                                                                                                   |     |
| ヘッダ | <machine.h>または<umachine.h>                                                                                                                                                               |     |
| 引数  | vec1                                                                                                                                                                                     | ベクタ |
|     | vec2                                                                                                                                                                                     | ベクタ |
|     | vec3                                                                                                                                                                                     | ベクタ |
| 例   | <pre>#include &lt;machine.h&gt; extern float data1[4]; extern float data2[4]; extern float data3[4]; void main(void) {     add4(data1,data2,data3);      /* data3=data1+data2 */ }</pre> |     |
| 備考  | cpu=sh4 を指定した場合のみ使用可能です。                                                                                                                                                                 |     |

## 4次元ベクタの差

***void sub4(float vec1[4], float vec2[4], float vec3[4])***

---

|     |                                                                                                                                                                                         |     |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 説明  | vec1(ベクタ)とvec2(ベクタ)の差をvec3(ベクタ)に格納します。                                                                                                                                                  |     |
| ヘッダ | <machine.h>または<umachine.h>                                                                                                                                                              |     |
| 引数  | vec1                                                                                                                                                                                    | ベクタ |
|     | vec2                                                                                                                                                                                    | ベクタ |
|     | vec3                                                                                                                                                                                    | ベクタ |
| 例   | <pre>#include &lt;machine.h&gt; extern float data1[4]; extern float data2[4]; extern float data3[4]; void main(void) {     sub4(data1,data2,data3);     /* data3=data1-data2 */ }</pre> |     |
| 備考  | cpu=sh4 を指定した場合のみ使用可能です。                                                                                                                                                                |     |

## 4×4 行列の乗算

***void mtrx4mul(float mat1[4], float mat2[4])***

説明 mat1(4×4 行列)を tbl(4×4 行列)で変換した結果 mat2 に格納します。  
tbl は組み込み関数 ld\_ext() でロードした行列です。

ヘッダ <machine.h>または<umachine.h>

引数 mat1                   4×4 行列  
mat2                   4×4 行列

例

```
#include <machine.h>
extern float tbl[4][4];
extern float tbl1[4][4];
extern float tbl2[4][4];
void main(void)
{
 ld_ext(tbl);
 mtrx4mul(tbl1,tbl2); /* tbl2=tbl1×tbl */
}
```

備考 cpu=sh4 を指定した場合のみ使用可能です。  
本関数は行列の演算のため非可換です。

例:

```
extern float matA[][];
extern float matB[][];
int judge(){
 float data1[4][4], data2[4][4];
 set_imask(15);
 ld_ext(matA);
 mtrx4mul(matB,data1);/* data1=matB×matA */
 ld_ext(matB);
 mtrx4mul(matA,data2);/* data2=matA×matB */
 /* この時の data1[][]と data2[][]の各要素は必ずしも一致しません。 */
}
```

## 4×4 行列の乗算と和

***void mtrx4muladd(float mat1[4], float mat2[4], float mat3[4])***

- 説明 mat1(4×4 行列)を tbl(4×4 行列)で変換した結果と mat2(4×4 行列)との和を mat3 に格納します。tbl は組み込み関数 ld\_ext() でロードした行列です。
- ヘッダ <machine.h>または<umachine.h>
- 引数 mat1 4×4 行列  
mat2 4×4 行列  
mat3 4×4 行列
- 例
- ```
#include <machine.h>
extern float tbl[4][4];
extern float tbl1[4][4];
extern float tbl2[4][4];
extern float tbl3[4][4];
void main(void)
{
    ld_ext(tbl);
    mtrx4muladd(tbl1,tbl2,tbl3); /* tbl3=tbl1×tbl +tbl2 */
}
```
- 備考 cpu=sh4 を指定した場合のみ使用可能です。
本関数は行列の演算のため非可換です。

4×4 行列の乗算と差

void mtrx4mulsub(float mat1[4], float mat2[4], float mat3[4])

- 説明 mat1(4×4 行列)を tbl(4×4 行列)で変換した結果と mat2(4×4 行列)との差を mat3 に格納します。tbl は組み込み関数 ld_ext() でロードした行列です。
- ヘッダ <machine.h>または<umachine.h>
- 引数 mat1 4×4 行列
mat2 4×4 行列
mat3 4×4 行列
- 例
- ```
#include <machine.h>
extern float tbl[4][4];
extern float tbl1[4][4];
extern float tbl2[4][4];
extern float tbl3[4][4];
void main(void)
{
 ld_ext(tbl);
 mtrx4mulsub(tbl1,tbl2,tbl3); /* tbl3=tbl1×tbl -tbl2 */
}
```
- 備考 cpu=sh4 を指定した場合のみ使用可能です。  
本関数は行列の演算のため非可換です。

**拡張レジスタへのロード*****void ld\_ext(float mat[4][4])***

説明 mat (4 × 4 行列) を拡張レジスタにロードします。

ヘッダ <machine.h>または<umachine.h>

引数 mat 4 × 4 行列

```
例
#include <machine.h>
extern float tbl[4][4];
void main(void)
{
 ld_ext(tbl);
}
```

備考 cpu=sh4 を指定した場合のみ使用可能です。  
本関数は、浮動小数点ステータス制御レジスタ (FPSCR) の浮動小数点レジスタバンクビット (FR) を変更して拡張レジスタにアクセスします。割り込み関数内で本関数を使用しているときは、ベクタ演算組み込み関数の前後で割り込みマスクを変更してください。

**拡張レジスタからのストア*****void st\_ext(float mat[4][4])***

説明 拡張レジスタの内容を mat (4 × 4 行列) にストアします。

ヘッダ <machine.h>または<umachine.h>

引数 mat 4 × 4 行列

```
例
#include <machine.h>
extern float tbl[4][4];
void main(void)
{
 st_ext(tbl);
}
```

備考 cpu=sh4 を指定した場合のみ使用可能です。  
本関数は、浮動小数点ステータス制御レジスタ (FPSCR) の浮動小数点レジスタバンクビット (FR) を変更して拡張レジスタにアクセスします。割り込み関数内で本関数を使用しているときは、ベクタ演算組み込み関数の前後で割り込みマスクを変更してください。

## 10.3 C/C++ライブラリ

### 10.3.1 標準Cライブラリ

#### (1) ライブラリの概要

C/C++言語の中で標準的に利用できるCライブラリ関数の仕様について説明します。ここでは、ライブラリの構成を概説し、本節の読み方および用語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ライブラリ関数の仕様を説明します。

#### (a) ライブラリの種類

ライブラリとは、入出力、文字列操作等の標準的な処理をC/C++言語の関数の形式で実現したものです。また、これらのライブラリは、各処理単位ごとに対応した標準インクルードファイルを取り込むことによって使用可能となります。

標準インクルードファイルには、対応するライブラリの宣言とそれらを使用するために必要なマクロ名が定義されています。

表 10.27 に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 10.27 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><br><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>            |

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 10.28 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 10.28 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 <float.h>  | 浮動小数点数の内部表現に関する各種制限値を定義します。               |
| 3 <limits.h> | コンパイラの内部処理に関する各種制限値を定義します。                |
| 4 <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。 |

## (b) ライブラリの説明形式

ライブラリの各関数を標準インクルードファイルごとに分類し、その標準インクルードファイルごとに説明します。その各分類は、まず、標準インクルードファイルの中で定義されているマクロ名や関数宣言に対する説明を行い(図 10.3 参照)、その後、各関数ごとの説明を行う(図 10.4 参照)という形式で構成されています。

図 10.3 に標準インクルードファイルの説明形式、図 10.4 に関数の説明形式を示します。

|                                                                                                                                                                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>項番 &lt;標準インクルードファイル名&gt;</p> <ul style="list-style-type: none"> <li>・本インクルードファイルがもつ全体的な機能の概要を説明します。</li> <li>・本インクルードファイル内で定義・宣言される名前を名前種別(【型】、【定数】、【変数】、【関数】)に分類して説明します。マクロである場合、名前種別のタイトル(【】内)または名前の説明箇所に(マクロ)と表記しています。</li> <li>・処理系定義仕様がある場合や、本インクルードファイル内で宣言されている関数に共通する注意事項がある場合、説明を補足します。</li> </ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

図 10.3 標準インクルードファイルの説明形式

|                                    |                                                                                                                                 |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>機能の概要を示します。</b>                 |                                                                                                                                 |
| <b>ライブラリ関数の型(リターン値および引数)を示します。</b> |                                                                                                                                 |
| 説明                                 | ライブラリ関数の機能を説明します。                                                                                                               |
| ヘッダ                                | 宣言元の標準インクルードファイル名です。                                                                                                            |
| リターン値                              | 正常：ライブラリ関数が正常終了したときの値です。<br>異常：ライブラリ関数が異常終了したときの値です。                                                                            |
| 引数                                 | 引数の意味を説明します。                                                                                                                    |
| 例                                  | 呼び出し手順を説明します。                                                                                                                   |
| エラー条件                              | ライブラリ関数の処理でリターン値からでは、判断できないエラーが発生する条件を示します。このようなエラーが発生したとき、エラーの種類に対応する、コンパイラごとに定義された値が <code>errno</code> <sup>*</sup> に設定されます。 |
| 備考                                 | 補足説明、または使用上の注意事項です。                                                                                                             |
| 処理系定義仕様                            | 本コンパイラの処理方法です。                                                                                                                  |

図 10.4 関数の説明形式

【注】\* `errno` は、ライブラリ関数実行中に生じたエラーの種類を格納する変数です。詳細については「10.3.1(2) `<stddef.h>`」を参照してください。

## (c) ライブラリ関数の説明で使用する用語

## (i) 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の機能呼び出ししていたのでは、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を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を行うことにより、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文字単位ごと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でできる機能をストリーム入出力といいます。

## (ii) 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。

## (iii) 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、その効果は保証されません。

例：

パラメ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。

## (iv) 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。



## (v) 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。

## (vi) ヌル文字

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。

## (vii) 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。

## (viii) 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。

## (ix) 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。

## (x) 浮動小数点数

浮動小数点数は、実数を近似して表現したものです。C言語のソースプログラム上では浮動小数点数を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数の表現は次のようになります。

$$2^n \times m \quad (n: \text{整数}, m: 2 \text{進小数})$$

ここで  $n$  を浮動小数点数の指数部、 $m$  を仮数部といいます。浮動小数点数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、浮動小数点数に関する用語を説明します。

- 基数

浮動小数点数が何進法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

浮動小数点数よりも精度の高い演算の途中結果を浮動小数点数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

浮動小数点数を、 $2^n \times m$ の形式で表現する場合、同一の数値を表わす異なる表現が可能です。

例：

$$2^5 \times 1.0_{(2)} \quad (_{(2)} \text{は} 2 \text{進数を示します})$$

$$2^6 \times 0.1_{(2)}$$

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数といい、浮動小数点数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

浮動小数点数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数よりも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。

## (xi) 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 10.29 に示す 12 種類があります。

表 10.29 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。   |

## (xii) 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。

## (xiii) 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。

## (xiv) 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。

## (d) 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、動作は保証されません。

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、動作は保証されません。

## 10. C/C++言語仕様

---

### (2) <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           | <code>ptrdiff_t</code> | 二つのポインタを減算した結果の型です。                                                                                                                                                                                                        |
| (マクロ)       | <code>size_t</code>    | <code>sizeof</code> 演算子による演算結果の型です。                                                                                                                                                                                        |
| 定数<br>(マクロ) | <code>NULL</code>      | ポインタが何も指していない時の値です。<br>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<br>(マクロ) | <code>errno</code>     | 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 <code>errno</code> に設定されます。<br>ライブラリ関数を呼び出す前に <code>errno</code> に 0 を設定しておき、ライブラリ関数の処理終了後に <code>errno</code> に設定されているコードを調べる<br>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。 |
| マクロ         | <code>offsetof</code>  | 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。                                                                                                                                                                                          |

### 処理系定義仕様

| 項目 |                                |                                   |
|----|--------------------------------|-----------------------------------|
| 1  | マクロ <code>NULL</code> の値       | <code>void</code> 型へのポインタ型の値 0 です |
| 2  | マクロ <code>ptrdiff_t</code> の内容 | <code>int</code> 型                |

- (3) `<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<br>(マクロ) | <code>assert</code> | 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されません。

## 診断

### *`void assert(int expression)`*

|     |                                                                                                                                                                                                                                         |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 説明  | プログラム中に診断機能を付け加えます。                                                                                                                                                                                                                     |
| ヘッダ | <code>&lt;assert.h&gt;</code>                                                                                                                                                                                                           |
| 引数  | <code>expression</code> 評価する式                                                                                                                                                                                                           |
| 例   | <pre>#include &lt;assert.h&gt; int expression;     assert (expression);</pre>                                                                                                                                                           |
| 備考  | <p><code>assert</code> マクロは、<code>expression</code> が真の時は値を返さずに処理を終了します。<code>expression</code> 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 <code>abort</code> 関数を呼び出します。</p> <p>診断情報の中には、パラメ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。</p> |

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。  
 ASSERTION FAILED: 式 FILE <ファイル名>,line <行番号>

## 10. C/C++言語仕様

### (4) <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。           |
|    | isctrl   | 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。        |

上記の関数において、入力パラメタの値が 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されません。

文字の種類の一覧を表 10.30 に示します。

表 10.30 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。<br>'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',<br>'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。<br>'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',<br>'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。<br>'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。<br>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。<br>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。<br>'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',<br>'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。                                                                                                                      |

## 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、isupper 関数で判定される文字集合 | unsigned char 型で表現できる文字集合です。判定の結果、真になる文字を表 10.31 に示します。 |

表 10.31 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'                       |

## 英字、10進数字判定

***int isalnum(int c)***

説 明 文字が英字または 10 進数字であるかどうか判定します。

ヘッダ <ctype.h>

リターン値 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

引 数 c 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=isalnum(c);
```

---

***int isalpha(int c)***

---

説明 文字が英字であるかどうか判定します。

ヘッダ <ctype.h>

リターン値 文字 *c* が英字の時 : 0 以外  
文字 *c* が英字以外の時 : 0

引数 *c* 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=isalpha(c);
```

---

***int iscntrl(int c)***

---

説明 文字が制御文字であるかどうか判定します。

ヘッダ <ctype.h>

リターン値 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

引数 *c* 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=iscntrl(c);
```



## 10 進数字判定

***int isdigit(int c)***

---

|       |                                                                 |
|-------|-----------------------------------------------------------------|
| 説明    | 文字が10進数字であるかどうか判定します。                                           |
| ヘッダ   | <ctype.h>                                                       |
| リターン値 | 文字 <i>c</i> が10進数字の時 : 0以外<br>文字 <i>c</i> が10進数字以外の時 : 0        |
| 引数    | <i>c</i> 判定する文字                                                 |
| 例     | <pre>#include &lt;ctype.h&gt; int c, ret; ret=isdigit(c);</pre> |

## 空白を除く印字文字判定

***int isgraph(int c)***

---

|       |                                                                  |
|-------|------------------------------------------------------------------|
| 説明    | 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。                                  |
| ヘッダ   | <ctype.h>                                                        |
| リターン値 | 文字 <i>c</i> が空白を除く印字文字の時 : 0以外<br>文字 <i>c</i> が空白を除く印字文字以外の時 : 0 |
| 引数    | <i>c</i> 判定する文字                                                  |
| 例     | <pre>#include &lt;ctype.h&gt; int c, ret; ret=isgraph(c);</pre>  |

---

***int islower(int c)***

---

説明 文字が英小文字であるかどうか判定します。

ヘッダ <ctype.h>

リターン値 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

引数 *c* 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=islower(c);
```

---

***int isprint(int c)***

---

説明 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。

ヘッダ <ctype.h>

リターン値 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

引数 *c* 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=isprint(c);
```

**特殊文字判定*****int ispunct(int c)***

説明 文字が特殊文字であるかどうか判定します。

ヘッダ <ctype.h>

リターン値 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

引数 *c* 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=ispunct(c);
```

**空白類文字判定*****int isspace(int c)***

説明 文字が空白類文字であるかどうか判定します。

ヘッダ <ctype.h>

リターン値 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

引数 *c* 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=isspace(c);
```

---

***int isupper(int c)***

---

説明 文字が英大文字であるかどうか判定します。

ヘッダ <ctype.h>

リターン値 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

引数 *c* 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=isupper(c);
```

---

***int isxdigit(int c)***

---

説明 文字が16進数字かどうか判定します。

ヘッダ <ctype.h>

リターン値 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

引数 *c* 判定する文字

例 

```
#include <ctype.h>
int c, ret;
ret=isxdigit(c);
```

**英小文字変換*****int tolower(int c)***

説明 英大文字を対応する英小文字に変換します。

ヘッダ <ctype.h>

リターン値 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*

引数 *c* 変換する文字

例  

```
#include <ctype.h>
int c, ret;
ret=tolower(c);
```

**英大文字変換*****int toupper(int c)***

説明 英小文字を対応する英大文字に変換します。

ヘッダ <ctype.h>

リターン値 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*

引数 *c* 変換する文字

例  

```
#include <ctype.h>
int c, ret;
ret=toupper(c);
```

## 10. C/C++言語仕様

### (5) <float.h>

浮動小数点数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    | FLT_RADIX       | 2                       | 指数部表現における基数です。                                                                                                                                                                            |
| (マクロ) | FLT_ROUNDS      | 1                       | 加算演算結果を丸めるかどうかを示します。本マクロの定義の意味は以下のとおりです。<br><ul style="list-style-type: none"> <li>・演算結果を丸める場合 : 正の値</li> <li>・演算結果を切り捨てる場合 : 0</li> <li>・特に規定しない場合 : -1</li> </ul> 丸め、切り捨てるの方法は、処理系定義です。 |
|       | FLT_GUARD       | 1                       | 乗算演算結果においてガードビットを用いるかどうかを示します。本マクロの定義の意味は以下のとおりです。<br><ul style="list-style-type: none"> <li>・ガードビットを用いる場合 : 1</li> <li>・ガードビットを用いない場合 : 0</li> </ul>                                     |
|       | FLT_NORMALIZE   | 1                       | 浮動小数点数の値を正規化するかどうかを示します。本マクロの定義の意味は以下のとおりです。<br><ul style="list-style-type: none"> <li>・正規化する場合 : 1</li> <li>・正規化しない場合 : 0</li> </ul>                                                     |
|       | FLT_MAX         | 3.4028235677973364e+38F | float 型が浮動小数点数値として表現できる最大値です。                                                                                                                                                             |
|       | DBL_MAX         | 1.7976931348623158e+308 | double 型が浮動小数点数値として表現できる最大値です。                                                                                                                                                            |
|       | LDBL_MAX        | 1.7976931348623158e+308 | long double 型が浮動小数点数値として表現できる最大値です。                                                                                                                                                       |
|       | FLT_MAX_EXP     | 127                     | float 型が浮動小数点数値として表現できる基数のべき乗の最大値です。                                                                                                                                                      |
|       | DBL_MAX_EXP     | 1023                    | double 型が浮動小数点数値として表現できる基数のべき乗の最大値です。                                                                                                                                                     |
|       | LDBL_MAX_EXP    | 1023                    | long double 型が浮動小数点数値として表現できる基数のべき乗の最大値です。                                                                                                                                                |
|       | FLT_MAX_10_EXP  | 38                      | float 型が浮動小数点数値として表現できる 10 のべき乗の最大値です。                                                                                                                                                    |
|       | DBL_MAX_10_EXP  | 308                     | double 型が浮動小数点数値として表現できる 10 のべき乗の最大値です。                                                                                                                                                   |
|       | LDBL_MAX_10_EXP | 308                     | long double 型が浮動小数点数値として表現できる 10 のべき乗の最大値です。                                                                                                                                              |
|       | FLT_MIN         | 1.175494351e-38F        | float 型が浮動小数点数値として表現できる正の値での最小値です。                                                                                                                                                        |
|       | DBL_MIN         | 2.2250738585072014e-308 | double 型が浮動小数点数値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 型が浮動小数点数値として表現できる正の値での最小値です。                                                                                                                                                  |
|       | FLT_MIN_EXP     | -149                    | float 型が正の値として表現できる浮動小数点数値の基数のべき乗の最小値です。                                                                                                                                                  |
|       | DBL_MIN_EXP     | -1074                   | double 型が正の値として表現できる浮動小数点数値の基数のべき乗の最小値です。                                                                                                                                                 |

| 種別               | 定義名             | 定義値                                                                | 説明                                                           |
|------------------|-----------------|--------------------------------------------------------------------|--------------------------------------------------------------|
| 定数<br>(マクロ)      | LDBL_MIN_EXP    | -1074                                                              | long double 型が正の値として表現できる浮動小数点数値の基数のべき乗の最小値です。               |
|                  | FLT_MIN_10_EXP  | -44                                                                | float 型が正の値として表現できる浮動小数点数値の 10 のべき乗の最小値です。                   |
|                  | DBL_MIN_10_EXP  | -323                                                               | double 型が正の値として表現できる浮動小数点数値の 10 のべき乗の最小値です。                  |
|                  | LDBL_MIN_10_EXP | -323                                                               | long double 型が正の値として表現できる浮動小数点数値の 10 のべき乗の最小値です。             |
|                  | FLT_DIG         | 6                                                                  | float 型の浮動小数点数値の 10 進精度の最大桁数です。                              |
|                  | DBL_DIG         | 15                                                                 | double 型の浮動小数点数値の 10 進精度の最大桁数です。                             |
|                  | LDBL_DIG        | 15                                                                 | long double 型の浮動小数点数値の 10 進精度の最大桁数です。                        |
|                  | FLT_MANT_DIG    | 24                                                                 | float 型の浮動小数点数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 型の浮動小数点数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 型の浮動小数点数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。               |
|                  | FLT_EXP_DIG     | 8                                                                  | float 型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数です。                     |
|                  | DBL_EXP_DIG     | 11                                                                 | double 型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数です。                    |
|                  | LDBL_EXP_DIG    | 11                                                                 | long double 型の浮動小数点数値を基数に合わせて表現した時の指数部の最大桁数です。               |
|                  | FLT_POS_EPS     | 5.9604648328104311e-8F                                             | float 型において、 $1.0 + x$ $1.0$ である最小の浮動小数点数値 $x$ を示します。        |
|                  | DBL_POS_EPS     | 1.1102230246251567e-16                                             | double 型において、 $1.0 + x$ $1.0$ である最小の浮動小数点数値 $x$ を示します。       |
|                  | LDBL_POS_EPS    | 1.1102230246251567e-16                                             | long double 型において、 $1.0 + x$ $1.0$ である最小の浮動小数点数値 $x$ を示します。  |
|                  | FLT_NEG_EPS     | 2.9802324164052156e-8F                                             | float 型において、 $1.0 - x$ $1.0$ である最小の浮動小数点数値 $x$ を示します。        |
|                  | DBL_NEG_EPS     | 5.5511151231257834e-17                                             | double 型において、 $1.0 - x$ $1.0$ である最小の浮動小数点数値 $x$ を示します。       |
|                  | LDBL_NEG_EPS    | 5.5511151231257834e-17                                             | long double 型において、 $1.0 - x$ $1.0$ である最小の浮動小数点数値 $x$ を示します。  |
|                  | FLT_POS_EPS_EXP | -23                                                                | float 型において、 $1.0 + (\text{基数})^n$ $1.0$ となる最小の整数 $n$ を示します。 |
| DBL_POS_EPS_EXP  | -52             | double 型において、 $1.0 + (\text{基数})^n$ $1.0$ となる最小の整数 $n$ を示します。      |                                                              |
| LDBL_POS_EPS_EXP | -52             | long double 型において、 $1.0 + (\text{基数})^n$ $1.0$ となる最小の整数 $n$ を示します。 |                                                              |

## 10. C/C++言語仕様

---

| 種別          | 定義名              | 定義値 | 説明                                                               |
|-------------|------------------|-----|------------------------------------------------------------------|
| 定数<br>(マクロ) | FLT_NEG_EPS_EXP  | -24 | float 型において、 $1.0 - (\text{基数})^n$ 1.0 となる最小の整数 $n$ を示します。       |
|             | DBL_NEG_EPS_EXP  | -53 | double 型において、 $1.0 - (\text{基数})^n$ 1.0 となる最小の整数 $n$ を示します。      |
|             | LDBL_NEG_EPS_EXP | -53 | long double 型において、 $1.0 - (\text{基数})^n$ 1.0 となる最小の整数 $n$ を示します。 |



## (6) &lt;limits.h&gt;

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<br>(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。           |
|             | 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。           |
|             | 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。  |

## 10. C/C++言語仕様

---

### (7) <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<br>(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<br>(マクロ) | ERANGE  | 「12.3 標準ライブラリのエラーメッセージ」を参照してください。             |
|             | EDOM    |                                               |
|             | EDIV    |                                               |
|             | ESTRN   |                                               |
|             | PTRERR  |                                               |
|             | ECBASE  |                                               |
|             | ETLN    |                                               |
|             | EEXP    |                                               |
|             | EEXPN   |                                               |
|             | EFLOATO |                                               |
|             | EFLOATU |                                               |
|             | EDBLO   |                                               |
|             | EDBLU   |                                               |
|             | ELDBLO  |                                               |
|             | ELDBLU  |                                               |
|             | NOTOPN  |                                               |
|             | EBADF   |                                               |
|             | ECSPEC  |                                               |

---

- (8) <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<br>(マクロ) | EDOM                       | 関数に入力するパラメ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 型の値として表わせない時、あるいはオーバーフロー/アンダフローとなった時、errno に設定する値です。 |
|             | HUGE_VAL                   | 関数の計算結果がオーバーフローした時に、関数のリターン値として返す値です。                                |
| 関数          | acos                       | 浮動小数点数の逆余弦を計算します。                                                    |
|             | asin                       | 浮動小数点数の逆正弦を計算します。                                                    |
|             | atan                       | 浮動小数点数の逆正接を計算します。                                                    |
|             | atan2                      | 浮動小数点数どうしを除算した結果の値の逆正接を計算します。                                        |
|             | cos                        | 浮動小数点数のラジアン値の余弦を計算します。                                               |
|             | sin                        | 浮動小数点数のラジアン値の正弦を計算します。                                               |
|             | tan                        | 浮動小数点数のラジアン値の正接を計算します。                                               |
|             | cosh                       | 浮動小数点数の双曲線余弦を計算します。                                                  |
|             | sinh                       | 浮動小数点数の双曲線正弦を計算します。                                                  |
|             | tanh                       | 浮動小数点数の双曲線正接を計算します。                                                  |
|             | exp                        | 浮動小数点数の指数関数を計算します。                                                   |
|             | frexp                      | 浮動小数点数を[0.5,1.0)の値と2のべき乗の積に分解します。                                    |
|             | ldexp                      | 浮動小数点数と2のべき乗の乗算を計算します。                                               |
|             | log                        | 浮動小数点数の自然対数を計算します。                                                   |
|             | log10                      | 浮動小数点数の10を底とする対数を計算します。                                              |
|             | modf                       | 浮動小数点数を整数部分と小数部分に分解します。                                              |
|             | pow                        | 浮動小数点数のべき乗を計算します。                                                    |
|             | sqrt                       | 浮動小数点数の正の平方根を計算します。                                                  |
|             | ceil                       | 浮動小数点数の小数点以下を切り上げた整数値を求めます。                                          |
|             | fabs                       | 浮動小数点数の絶対値を計算します。                                                    |
|             | floor                      | 浮動小数点数の小数点以下を切り捨てた整数値を求めます。                                          |
| fmod        | 浮動小数点数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`型の値として表わ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、計算結果がオーバーフローの時は、正しく計算が行われた時と同様の符号の`HUGE_VAL`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、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。

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、浮動小数点数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。
3. `cpu=sh4`を指定し、かつ`fabs`関数および`sqrt`関数を使用した場合、もしくは`cpu=sh2e`かつ`double=float`を指定し、かつ`fabs`関数を使用した場合は、エラーが発生しても`errno`は設定されません。

### 処理系定義仕様

|   | 項目                                                                                 | コンパイラの仕様                                  |
|---|------------------------------------------------------------------------------------|-------------------------------------------|
| 1 | 数学関数の入力引数値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「10.1.3 浮動小数点数の仕様」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「 <code>ERANGE</code> 」の値が「 <code>errno</code> 」に設定されるかどうか | 設定しません。                                   |
| 3 | <code>fmod</code> 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。                               |

**逆余弦*****double acos(double d)***

|       |                                                                |               |
|-------|----------------------------------------------------------------|---------------|
| 説明    | 浮動小数点数の逆余弦を計算します。                                              |               |
| ヘッダ   | <code>&lt;math.h&gt;</code>                                    |               |
| リターン値 | 正常：dの逆余弦値<br>異常：定義域エラーの時は、非数を返します                              |               |
| 引数    | d                                                              | 逆余弦を求める浮動小数点数 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=acos(d);</pre> |               |
| エラー条件 | 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。                          |               |
| 備考    | acos 関数のリターン値の範囲は[0, ]です。                                      |               |

**逆正弦*****double asin(double d)***

|       |                                                                |               |
|-------|----------------------------------------------------------------|---------------|
| 説明    | 浮動小数点数の逆正弦を計算します。                                              |               |
| ヘッダ   | <code>&lt;math.h&gt;</code>                                    |               |
| リターン値 | 正常：dの逆正弦値<br>異常：定義域エラーの時は、非数を返します                              |               |
| 引数    | d                                                              | 逆正弦を求める浮動小数点数 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=asin(d);</pre> |               |
| エラー条件 | 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。                          |               |
| 備考    | asin 関数のリターン値の範囲は[- /2, /2]です。                                 |               |

---

***double atan(double d)***

---

説明 浮動小数点数の逆正接を計算します。

ヘッダ <math.h>

リターン値 d の逆正接値

引数 d 逆正接を求める浮動小数点数

例 

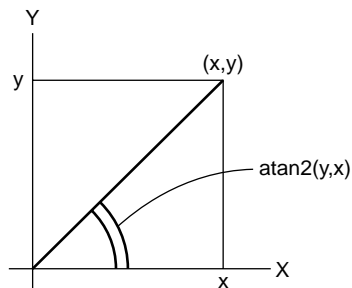
```
#include <math.h>
double d, ret;
ret=atan(d);
```

備考 atan 関数のリターン値の範囲は(- /2, /2)です。

## 除算後の逆正接

***double atan2(double y, double x)***

|       |                                                                                                                                                                                                                                      |     |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 説明    | 浮動小数点数どうしを除算した結果の値の逆正接を計算します。                                                                                                                                                                                                        |     |
| ヘッダ   | <math.h>                                                                                                                                                                                                                             |     |
| リターン値 | 正常： $y$ を $x$ で除算したときの逆正接値<br>異常：定義域エラーの時は、非数を返します                                                                                                                                                                                   |     |
| 引数    | $x$                                                                                                                                                                                                                                  | 除数  |
|       | $y$                                                                                                                                                                                                                                  | 被除数 |
| 例     | <pre>#include &lt;math.h&gt; double x, y, ret; ret=atan2(y,x);</pre>                                                                                                                                                                 |     |
| エラー条件 | $x, y$ の値がともに $0.0$ の時、定義域エラーになります。                                                                                                                                                                                                  |     |
| 備考    | $\text{atan2}$ 関数のリターン値の範囲は $(-\pi, \pi]$ です。 $\text{atan2}$ 関数の示す意味を図 10.5 に示します。図に示すように、 $\text{atan2}$ 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 $\pi$ 、 $x=0.0$ の時、 $y$ の値の正負に従って結果は $\pm \pi/2$ となります。 |     |

図 10.5  $\text{atan2}$  関数の意味

***double cos(double d)***

---

説明 浮動小数点数のラジアン値の余弦を計算します。

ヘッダ <math.h>

リターン値 dの余弦値

引数 d 余弦を求めるラジアン値

例 

```
#include <math.h>
double d, ret;
ret=cos(d);
```

***double sin(double d)***

---

説明 浮動小数点数のラジアン値の正弦を計算します。

ヘッダ <math.h>

リターン値 dの正弦値

引数 d 正弦を求めるラジアン値

例 

```
#include <math.h>
double d, ret;
ret=sin(d);
```



**正接*****double tan(double d)***

---

|       |                                                               |             |
|-------|---------------------------------------------------------------|-------------|
| 説明    | 浮動小数点数のラジアン値の正接を計算します。                                        |             |
| ヘッダ   | <math.h>                                                      |             |
| リターン値 | dの正接値                                                         |             |
| 引数    | d                                                             | 正接を求めるラジアン値 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=tan(d);</pre> |             |

**双曲線余弦*****double cosh(double d)***

---

|       |                                                                |                 |
|-------|----------------------------------------------------------------|-----------------|
| 説明    | 浮動小数点数の双曲線余弦を計算します。                                            |                 |
| ヘッダ   | <math.h>                                                       |                 |
| リターン値 | dの双曲線余弦値                                                       |                 |
| 引数    | d                                                              | 双曲線余弦を求める浮動小数点数 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=cosh(d);</pre> |                 |

---

***double sinh(double d)***

---

説明 浮動小数点数の双曲線正弦を計算します。

ヘッダ <math.h>

リターン値 dの双曲線正弦値

引数 d 双曲線正弦を求める浮動小数点数

例 

```
#include <math.h>
double d, ret;
ret=sinh(d);
```

---

***double tanh(double d)***

---

説明 浮動小数点数の双曲線正接を計算します。

ヘッダ <math.h>

リターン値 dの双曲線正接値

引数 d 双曲線正接を求める浮動小数点数

例 

```
#include <math.h>
double d, ret;
ret=tanh(d);
```

## 指数関数

***double exp(double d)***

|       |                                                               |                |
|-------|---------------------------------------------------------------|----------------|
| 説明    | 浮動小数点数の指数関数を計算します。                                            |                |
| ヘッダ   | <math.h>                                                      |                |
| リターン値 | d の指数関数値                                                      |                |
| 引数    | d                                                             | 指数関数を求める浮動小数点数 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=exp(d);</pre> |                |

## 浮動小数点数を仮数、指数に分解

***double frexp(double value, int \*e)***

|       |                                                                                                                                                                                                         |                                                         |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| 説明    | 浮動小数点数を [0.5, 1.0) の値と 2 のべき乗の積に分解します。                                                                                                                                                                  |                                                         |
| ヘッダ   | <math.h>                                                                                                                                                                                                |                                                         |
| リターン値 | value が 0.0 の時                                                                                                                                                                                          | : 0.0                                                   |
|       | value が 0.0 でない時                                                                                                                                                                                        | : ret * 2 <sup>e</sup> の指している領域の値 = value で定義される ret の値 |
| 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数                      |
|       | e                                                                                                                                                                                                       | 2 のべき乗値を格納する記憶域へのポインタ                                   |
| 例     | <pre>#include &lt;math.h&gt; double ret, value; int *e; ret=frexp(value,e);</pre>                                                                                                                       |                                                         |
| 備考    | <p>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。e 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。</p> <p>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。</p> <p>value が 0.0 ならば、e の指す int 型の記憶域の内容と ret の値は 0.0 になります。</p> |                                                         |

---

***double ldexp(double e, int f)***

---

|       |                                                                           |                  |
|-------|---------------------------------------------------------------------------|------------------|
| 説明    | 浮動小数点数と2のべき乗の積を計算します。                                                     |                  |
| ヘッダ   | <math.h>                                                                  |                  |
| リターン値 | $e * 2^f$ の演算結果の値                                                         |                  |
| 引数    | e                                                                         | 2のべき乗値を求める浮動小数点数 |
|       | f                                                                         | 2のべき乗値           |
| 例     | <pre>#include &lt;math.h&gt; double ret, e; int f; ret=ldexp(e, f);</pre> |                  |

---

***double log(double d)***

---

|       |                                                               |                |
|-------|---------------------------------------------------------------|----------------|
| 説明    | 浮動小数点数の自然対数を計算します。                                            |                |
| ヘッダ   | <math.h>                                                      |                |
| リターン値 | 正常:dの自然対数の値<br>異常:定義域エラーの時は、非数を返します                           |                |
| 引数    | d                                                             | 自然対数を求める浮動小数点数 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=log(d);</pre> |                |
| エラー条件 | dの値が負の時、定義域エラーになります。<br>dの値が0.0の時、範囲エラーになります。                 |                |

## 常用対数

***double log10(double d)***

|       |                                                                 |                      |
|-------|-----------------------------------------------------------------|----------------------|
| 説明    | 浮動小数点数の 10 を底とする対数を計算します。                                       |                      |
| ヘッダ   | <math.h>                                                        |                      |
| リターン値 | 正常 : d は 10 を底とする対数値<br>異常 : 定義域エラーの時は、非数を返します                  |                      |
| 引数    | d                                                               | 10 を底とする対数を求める浮動小数点数 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=log10(d);</pre> |                      |
| エラー条件 | d の値が負の値の時、定義域エラーになります。<br>d の値が 0.0 の時、範囲エラーになります。             |                      |

## 浮動小数点数を整数部、小数部に分解

***double modf(double a, double \*b)***

|       |                                                                      |                                             |
|-------|----------------------------------------------------------------------|---------------------------------------------|
| 説明    | 浮動小数点数を整数部分と小数部分に分解します。                                              |                                             |
| ヘッダ   | <math.h>                                                             |                                             |
| リターン値 | a の小数部分                                                              |                                             |
| 引数    | a<br>b                                                               | 整数部分と小数部分に分解する浮動小数点数<br>整数部分を格納する記憶域を指すポインタ |
| 例     | <pre>#include &lt;math.h&gt; double a, *b, ret; ret=modf(a,b);</pre> |                                             |

---

***double pow(double x, double y)***

---

|       |                                                                      |         |
|-------|----------------------------------------------------------------------|---------|
| 説明    | 浮動小数点数のべき乗を計算します。                                                    |         |
| ヘッダ   | <math.h>                                                             |         |
| リターン値 | 正常：x の y 乗の値<br>異常：定義域エラーの時は、非数を返します                                 |         |
| 引数    | x                                                                    | べき乗される値 |
|       | y                                                                    | べき乗する値  |
| 例     | <pre>#include &lt;math.h&gt; double x, y, ret; ret=pow(x,y);</pre>   |         |
| 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。 |         |

---

***double sqrt(double d)***

---

|       |                                                                |                 |
|-------|----------------------------------------------------------------|-----------------|
| 説明    | 浮動小数点数の正の平方根を計算します。                                            |                 |
| ヘッダ   | <math.h>                                                       |                 |
| リターン値 | 正常：d の正の平方根の値<br>異常：定義域エラーの時は、非数を返します                          |                 |
| 引数    | d                                                              | 正の平方根を求める浮動小数点数 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=sqrt(d);</pre> |                 |
| エラー条件 | d の値が負の値の時、定義域エラーになります。                                        |                 |

**切り上げ*****double ceil(double d)***

説明 浮動小数点数の小数点以下を切り上げた整数値を求めます。

ヘッダ <math.h>

リターン値 d の小数点以下を切り上げた整数値

引数 d 小数点以下を切り上げる浮動小数点数

例  

```
#include <math.h>
double d, ret;
ret=ceil(d);
```

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。

**絶対値*****double fabs(double d)***

説明 浮動小数点数の絶対値を計算します。

ヘッダ <math.h>

リターン値 d の絶対値

引数 d 絶対値を求める浮動小数点数

例  

```
#include <math.h>
double d, ret;
ret=fabs(d);
```

***double floor(double d)***

|       |                                                                                         |                   |
|-------|-----------------------------------------------------------------------------------------|-------------------|
| 説明    | 浮動小数点数の小数点以下を切り捨てた整数値を求めます。                                                             |                   |
| ヘッダ   | <math.h>                                                                                |                   |
| リターン値 | d の小数点以下を切り捨てた整数値                                                                       |                   |
| 引数    | d                                                                                       | 小数点以下を切り捨てる浮動小数点数 |
| 例     | <pre>#include &lt;math.h&gt; double d, ret; ret=floor(d);</pre>                         |                   |
| 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。 |                   |

***double fmod(double x, double y)***

|       |                                                                                                                                                      |     |
|-------|------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 説明    | 浮動小数点数どうしを除算した結果の余りを計算します。                                                                                                                           |     |
| ヘッダ   | <math.h>                                                                                                                                             |     |
| リターン値 | y の値が 0.0 の時 : x<br>y の値が 0.0 でない時 : x を y で除算した結果の余り                                                                                                |     |
| 引数    | x                                                                                                                                                    | 被除数 |
|       | y                                                                                                                                                    | 除数  |
| 例     | <pre>#include &lt;math.h&gt; double x, y, ret; ret=fmod(x,y);</pre>                                                                                  |     |
| 備考    | fmod 関数では、パラメタ x, y、リターン値 ret の間には、次に示す関係が成立します。<br>$x=y*i+ret$ (ただし i は整数値)<br>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。<br>x/y の商を表現できない場合、結果の値は、保証されません。 |     |



## (9) &lt;mathf.h&gt;

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<br>(マクロ) | EDOM                        | 関数に入力するパラメ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 型の値として表わせない時、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                    | 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                       | 浮動小数点数の逆余弦を計算します。                                                    |
|             | asinf                       | 浮動小数点数の逆正弦を計算します。                                                    |
|             | atanf                       | 浮動小数点数の逆正接を計算します。                                                    |
|             | atan2f                      | 浮動小数点数どうしを除算した結果の値の逆正接を計算します。                                        |
|             | cosf                        | 浮動小数点数のラジアン値の余弦を計算します。                                               |
|             | sinf                        | 浮動小数点数のラジアン値の正弦を計算します。                                               |
|             | tanf                        | 浮動小数点数のラジアン値の正接を計算します。                                               |
|             | coshf                       | 浮動小数点数の双曲線余弦を計算します。                                                  |
|             | sinhf                       | 浮動小数点数の双曲線正弦を計算します。                                                  |
|             | tanhf                       | 浮動小数点数の双曲線正接を計算します。                                                  |
|             | expf                        | 浮動小数点数の指数関数を計算します。                                                   |
|             | frexpf                      | 浮動小数点数を[0.5, 1.0)の値と 2 のべき乗の積に分解します。                                 |
|             | ldexpf                      | 浮動小数点数と 2 のべき乗の乗算を計算します。                                             |
|             | logf                        | 浮動小数点数の自然対数を計算します。                                                   |
|             | log10f                      | 浮動小数点数の 10 を底とする対数を計算します。                                            |
|             | modff                       | 浮動小数点数を整数部分と小数部分に分解します。                                              |
|             | powf                        | 浮動小数点数のべき乗を計算します。                                                    |
|             | sqrtf                       | 浮動小数点数の正の平方根を計算します。                                                  |
|             | ceilf                       | 浮動小数点数の小数点以下を切り上げた整数値を求めます。                                          |
|             | fabsf                       | 浮動小数点数の絶対値を計算します。                                                    |
| floorf      | 浮動小数点数の小数点以下を切り捨てた整数値を求めます。 |                                                                      |
| fmodf       | 浮動小数点数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`型の値として表わ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、計算結果がオーバーフローの時は、正しく計算が行われた時と同様の符号の`HUGE_VAL`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、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。

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、浮動小数点数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。
3. `cpu=sh4` を指定し、かつ `fabsf` 関数および `sqrtf` 関数を使用した場合、もしくは `cpu=sh2e` を指定し、かつ `fabsf` 関数を使用した場合は、エラーが発生しても `errno` は設定されません。

### 処理系定義仕様

|   | 項目                                                                                 | コンパイラの仕様                                  |
|---|------------------------------------------------------------------------------------|-------------------------------------------|
| 1 | 数学関数の入力引数値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「10.1.3 浮動小数点数の仕様」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「 <code>ERANGE</code> 」の値が「 <code>errno</code> 」に設定されるかどうか | 設定しません。                                   |
| 3 | <code>fmodf</code> 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。                               |

**逆余弦*****float acosf(float f)***

|       |                                                                 |               |
|-------|-----------------------------------------------------------------|---------------|
| 説明    | 浮動小数点数の逆余弦を計算します。                                               |               |
| ヘッダ   | <code>&lt;mathf.h&gt;</code>                                    |               |
| リターン値 | 正常：f の逆余弦値<br>異常：定義域エラーの時は、非数を返します                              |               |
| 引数    | f                                                               | 逆余弦を求める浮動小数点数 |
| 例     | <pre>#include &lt;mathf.h&gt; float f, ret; ret=acosf(f);</pre> |               |
| エラー条件 | 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。                          |               |
| 備考    | acosf 関数のリターン値の範囲は[0, ]です。                                      |               |

**逆正弦*****float asinf(float f)***

|       |                                                                 |               |
|-------|-----------------------------------------------------------------|---------------|
| 説明    | 浮動小数点数の逆正弦を計算します。                                               |               |
| ヘッダ   | <code>&lt;mathf.h&gt;</code>                                    |               |
| リターン値 | 正常：f の逆正弦値<br>異常：定義域エラーの時は、非数を返します                              |               |
| 引数    | f                                                               | 逆正弦を求める浮動小数点数 |
| 例     | <pre>#include &lt;mathf.h&gt; float f, ret; ret=asinf(f);</pre> |               |
| エラー条件 | 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。                          |               |
| 備考    | asinf 関数のリターン値の範囲は[- /2, /2]です。                                 |               |

---

***float atanf(float f)***

---

説 明 浮動小数点数の逆正接を計算します。

ヘッダ <mathf.h>

リターン値 f の逆正接値

引 数 f 逆正接を求める浮動小数点数

例 

```
#include <mathf.h>
float f, ret;
ret=atanf(f);
```

備 考 atanf 関数のリターン値の範囲は(- /2, /2)です。

## 除算後の逆正接

***float atan2f(float y, float x)***

説明 浮動小数点数どうしを除算した結果の値の逆正接を計算します。

ヘッダ <mathf.h>

リターン値 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

引数  $x$  除数  
 $y$  被除数

例  

```
#include <mathf.h>
float x, y, ret;
ret=atan2f(y,x);
```

エラー条件  $x, y$  の値がともに  $0.0$  の時、定義域エラーになります。

備考  $\text{atan2f}$  関数のリターン値の範囲は  $(-\pi, \pi]$  です。 $\text{atan2f}$  関数の示す意味を図 10.6 に示します。図に示すように、 $\text{atan2f}$  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  $\pi$ 、 $x = 0.0$  の時、 $y$  の値の正負に従って結果は  $\pm \pi/2$  となります。

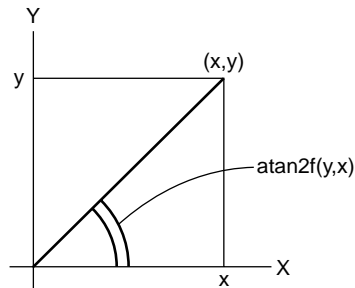


図 10.6  $\text{atan2f}$  関数の意味

---

***float cosf(float f)***

---

説明 浮動小数点数のラジアン値の余弦を計算します。

ヘッダ <mathf.h>

リターン値 f の余弦値

引数 f 余弦を求めるラジアン値

例 

```
#include <mathf.h>
float f, ret;
ret=cosf(f);
```

---

***float sinf(float f)***

---

説明 浮動小数点数のラジアン値の正弦を計算します。

ヘッダ <mathf.h>

リターン値 f の正弦値

引数 f 正弦を求めるラジアン値

例 

```
#include <mathf.h>
float f, ret;
ret=sinf(f);
```

**正接*****float tanf(float f)***

説明 浮動小数点数のラジアン値の正接を計算します。

ヘッダ <mathf.h>

リターン値 f の正接値

引数 f 正接を求めるラジアン値

例

```
#include <mathf.h>
float f, ret;
ret=tanf(f);
```

**双曲線余弦*****float coshf(float f)***

説明 浮動小数点数の双曲線余弦を計算します。

ヘッダ <mathf.h>

リターン値 f の双曲線余弦値

引数 f 双曲線余弦を求める浮動小数点数

例

```
#include <mathf.h>
float f, ret;
ret=coshf(f);
```

---

***float sinh(float f)***

---

説明 浮動小数点数の双曲線正弦を計算します。

ヘッダ <mathf.h>

リターン値 f の双曲線正弦値

引数 f 双曲線正弦を求める浮動小数点数

例 

```
#include <mathf.h>
float f, ret;
ret=sinh(f);
```

---

***float tanh(float f)***

---

説明 浮動小数点数の双曲線正接を計算します。

ヘッダ <mathf.h>

リターン値 f の双曲線正接値

引数 f 双曲線正接を求める浮動小数点数

例 

```
#include <mathf.h>
float f, ret;
ret=tanh(f);
```



## 指数関数

***float expf(float f)***

|       |                                                                |                |
|-------|----------------------------------------------------------------|----------------|
| 説 明   | 浮動小数点数の指数関数を計算します。                                             |                |
| ヘッダ   | <mathf.h>                                                      |                |
| リターン値 | f の指数関数値                                                       |                |
| 引 数   | f                                                              | 指数関数を求める浮動小数点数 |
| 例     | <pre>#include &lt;mathf.h&gt; float f, ret; ret=expf(f);</pre> |                |

## 浮動小数点数を仮数、指数に分解

***float frexpf(float value, int \*e)***

|       |                                                                                                                                                                                                          |                                                         |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| 説 明   | 浮動小数点数を [0.5, 1.0) の値と 2 のべき乗の積に分解します。                                                                                                                                                                   |                                                         |
| ヘッダ   | <mathf.h>                                                                                                                                                                                                |                                                         |
| リターン値 | value が 0.0 の時                                                                                                                                                                                           | : 0.0                                                   |
|       | value が 0.0 でない時                                                                                                                                                                                         | : ret * 2 <sup>e</sup> の指している領域の値 = value で定義される ret の値 |
| 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数                      |
|       | e                                                                                                                                                                                                        | 2 のべき乗値を格納する記憶域へのポインタ                                   |
| 例     | <pre>#include &lt;mathf.h&gt; float ret, value; int *e; ret=frexpf(value,e);</pre>                                                                                                                       |                                                         |
| 備 考   | <p>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。e 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。</p> <p>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。</p> <p>value が 0.0 ならば、e の指す int 型の記憶域の内容と ret の値は 0.0 になります。</p> |                                                         |

---

***float ldexpf(float e, int f)***

---

|       |                                                                            |                  |
|-------|----------------------------------------------------------------------------|------------------|
| 説明    | 浮動小数点数と2のべき乗の積を計算します。                                                      |                  |
| ヘッダ   | <mathf.h>                                                                  |                  |
| リターン値 | $e * 2^f$ の演算結果の値                                                          |                  |
| 引数    | e                                                                          | 2のべき乗値を求める浮動小数点数 |
|       | f                                                                          | 2のべき乗値           |
| 例     | <pre>#include &lt;mathf.h&gt; float ret, e; int f; ret=ldexpf(e, f);</pre> |                  |

---

***float logf(float f)***

---

|       |                                                                |                |
|-------|----------------------------------------------------------------|----------------|
| 説明    | 浮動小数点数の自然対数を計算します。                                             |                |
| ヘッダ   | <mathf.h>                                                      |                |
| リターン値 | 正常：fの自然対数の値<br>異常：定義域エラーの時は、非数を返します                            |                |
| 引数    | f                                                              | 自然対数を求める浮動小数点数 |
| 例     | <pre>#include &lt;mathf.h&gt; float f, ret; ret=logf(f);</pre> |                |
| エラー条件 | fの値が負の時、定義域エラーになります。<br>fの値が0.0の時、範囲エラーになります。                  |                |

## 常用対数

***float log10f(float f)***

|       |                                                                  |                      |
|-------|------------------------------------------------------------------|----------------------|
| 説明    | 浮動小数点数の 10 を底とする対数を計算します。                                        |                      |
| ヘッダ   | <mathf.h>                                                        |                      |
| リターン値 | 正常：f は 10 を底とする対数値<br>異常：定義域エラーの時は、非数を返します                       |                      |
| 引数    | f                                                                | 10 を底とする対数を求める浮動小数点数 |
| 例     | <pre>#include &lt;mathf.h&gt; float f, ret; ret=log10f(f);</pre> |                      |
| エラー条件 | f の値が負の値の時、定義域エラーになります。<br>f の値が 0.0 の時、範囲エラーになります。              |                      |

## 浮動小数点数を整数部、小数部に分解

***float modff(float a, float \*b)***

|       |                                                                       |                                             |
|-------|-----------------------------------------------------------------------|---------------------------------------------|
| 説明    | 浮動小数点数を整数部分と小数部分に分解します。                                               |                                             |
| ヘッダ   | <mathf.h>                                                             |                                             |
| リターン値 | a の小数部分                                                               |                                             |
| 引数    | a<br>b                                                                | 整数部分と小数部分に分解する浮動小数点数<br>整数部分を格納する記憶域を指すポインタ |
| 例     | <pre>#include &lt;mathf.h&gt; float a, *b, ret; ret=modff(a,b);</pre> |                                             |

---

***float powf(float x, float y)***

---

|       |                                                                                  |         |
|-------|----------------------------------------------------------------------------------|---------|
| 説明    | 浮動小数点数のべき乗を計算します。                                                                |         |
| ヘッダ   | <code>&lt;mathf.h&gt;</code>                                                     |         |
| リターン値 | 正常： $x$ の $y$ 乗の値<br>異常：定義域エラーの時は、非数を返します                                        |         |
| 引数    | $x$                                                                              | べき乗される値 |
|       | $y$                                                                              | べき乗する値  |
| 例     | <pre>#include &lt;mathf.h&gt; float x, y, ret; ret=powf(x, y);</pre>             |         |
| 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。 |         |

---

***float sqrtf(float f)***

---

|       |                                                                 |                 |
|-------|-----------------------------------------------------------------|-----------------|
| 説明    | 浮動小数点数の正の平方根を計算します。                                             |                 |
| ヘッダ   | <code>&lt;mathf.h&gt;</code>                                    |                 |
| リターン値 | 正常： $f$ の正の平方根の値<br>異常：定義域エラーの時は、非数を返します                        |                 |
| 引数    | $f$                                                             | 正の平方根を求める浮動小数点数 |
| 例     | <pre>#include &lt;mathf.h&gt; float f, ret; ret=sqrtf(f);</pre> |                 |
| エラー条件 | $f$ の値が負の値の時、定義域エラーになります。                                       |                 |

## 切り上げ

***float ceilf(float f)***

---

|       |                                                                                            |
|-------|--------------------------------------------------------------------------------------------|
| 説明    | 浮動小数点数の小数点以下を切り上げた整数値を求めます。                                                                |
| ヘッダ   | <mathf.h>                                                                                  |
| リターン値 | f の小数点以下を切り上げた整数値                                                                          |
| 引数    | f                                    小数点以下を切り上げる浮動小数点数                                     |
| 例     | <pre>#include &lt;mathf.h&gt; float f, ret; ret=ceilf(f);</pre>                            |
| 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。 |

## 絶対値

***float fabsf(float f)***

---

|       |                                                                 |
|-------|-----------------------------------------------------------------|
| 説明    | 浮動小数点数の絶対値を計算します。                                               |
| ヘッダ   | <mathf.h>                                                       |
| リターン値 | f の絶対値                                                          |
| 引数    | f                                    絶対値を求める浮動小数点数              |
| 例     | <pre>#include &lt;mathf.h&gt; float f, ret; ret=fabsf(f);</pre> |

***float floorf(float f)***

説明 浮動小数点数の小数点以下を切り捨てた整数値を求めます。

ヘッダ <mathf.h>

リターン値 f の小数点以下を切り捨てた整数値

引数 f 小数点以下を切り捨てる浮動小数点数

例

```
#include <mathf.h>
float f, ret;
ret=floorf(f);
```

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。

***float fmodf(float x, float y)***

説明 浮動小数点数どうしを除算した結果の余りを計算します。

ヘッダ <mathf.h>

リターン値 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

引数 x 被除数  
y 除数

例

```
#include <mathf.h>
float x, y, ret;
ret=fmodf(x, y);
```

備考 fmodf 関数では、パラメタ 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、保証されません。

- (10) <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<br>(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 main()
5 {
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 パラメタで指定した値(1)になります。その結果、9 行目以降が実行されます。

***int setjmp(jmp\_buf env)***

|       |                                                                                                                                                                                                                                                                             |                    |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| 説明    | 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。                                                                                                                                                                                                                                               |                    |
| ヘッダ   | <setjmp.h>                                                                                                                                                                                                                                                                  |                    |
| リターン値 | 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             |
| 引数    | env                                                                                                                                                                                                                                                                         | 実行環境を退避する記憶域へのポインタ |
| 例     | <pre>#include &lt;setjmp.h&gt; int ret; jmp_buf env; ret=setjmp(env);</pre>                                                                                                                                                                                                 |                    |
| 備考    | <p>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 パラメタの値となります。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。</p> |                    |

***void longjmp(jmp\_buf env, int ret)***

|     |                                                                                                                                                                                                                                                                                                                                   |                    |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| 説明  | 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。                                                                                                                                                                                                                                                                     |                    |
| ヘッダ | <setjmp.h>                                                                                                                                                                                                                                                                                                                        |                    |
| 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 |
| 例   | <pre>#include &lt;setjmp.h&gt; int ret; jmp_buf env; longjmp(env,ret);</pre>                                                                                                                                                                                                                                                      |                    |
| 備考  | <p>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 パラメタ 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 パラメタ 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。</p> <p>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 関数の動作は保証されません。</p> |                    |



## (11) &lt;stdarg.h&gt;

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<br>(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<br>(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。                                  |

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パラメタに出力するデータの数を指定し、以下の引数と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。その後引数の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パラメタに指定します。

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)。

## 可変個引数取り出し開始

***void va\_start(va\_list ap, parmN)***

説明 可変個のパラメタへの参照を行うため、初期設定処理を行います。

ヘッダ <stdarg.h>

引数 ap 可変個のパラメタにアクセスするための変数  
parmN 最右端の引数の識別子

例

```
#include <stdarg.h>
void va_list(int count, ...)
{
 va_list ap;
 va_start(ap, count);
}
```

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パラメタの並びの最右端のパラメタの識別子、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。

## 可変個引数取り出し

***type va\_arg(va\_list ap, type)***

説明 可変個のパラメタを持つ関数に対して、現在参照中のパラメタの次のパラメタへの参照を可能とします。

ヘッダ <stdarg.h>

リターン値 パラメタの値

引数 ap 可変個のパラメタにアクセスするための変数  
type アクセスするパラメタの型

例

```
#include <stdarg.h>
va_list ap;
int ret;
ret=va_arg(ap, int);
```

備考 va\_start マクロで初期化した va\_list 型の変数を第 1 パラメタに指定します。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パラメタが順次本マクロのリターン値として返されます。第 2 パラメタ 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、正しくパラメタ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されません。

**可変個引数取り出し終了*****void va\_end(va\_list ap)***

---

|     |                                                                                                        |
|-----|--------------------------------------------------------------------------------------------------------|
| 説明  | 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。                                                                             |
| ヘッダ | <stdarg.h>                                                                                             |
| 引数  | ap                                    可変個の引数を参照するための変数                                                 |
| 例   | <pre>#include &lt;stdarg.h&gt; va_list ap;     va_end(ap);</pre>                                       |
| 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。<br>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されません。 |

## 10. C/C++言語仕様

### (12) <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<br>(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な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小値です。                            |
|             | 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。 |                                                                 |
| 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。     |
|             | 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ヘデータを出力します。                                  |
|             | 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。                             |

| 種別 | 定義名      | 説明                                           |
|----|----------|----------------------------------------------|
| 関数 | 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。        |
|    | 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。        |
|    | 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。                |
|    | vfprintf | 可変個のパラメタ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。   |
|    | vprintf  | 可変個のパラメタ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。    |
|    | vsprintf | 可変個のパラメタ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文字を入力します。                    |
|    | 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文字を出力します。                     |
|    | 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文字を入力します。              |
|    | getchar  | (マクロ) 標準入力ファイルから1文字を入力します。                   |
|    | 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文字を出力します。               |
|    | putchar  | (マクロ) 標準出力ファイルへ1文字を出力します。                    |
|    | 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。 |

## 10. C/C++言語仕様

### 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。低水準インターフェースルーチンの仕様によります。                                            |
| 2  | 改行文字の直前にか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<br>fscanf 関数での変換文字「-」の意味 | 16 進数出力となります。<br>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。                    |
| 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。<br>ftell 関数については規定しません。<br>低水準インター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。                                                       |
| 14 | calloc,malloc,realloc 関数でサイズが0のときの動作               | 0 バイトの領域を割り付けます。                                                           |

(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 10.32 に示します。

表 10.32 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。

***int fclose(FILE \*fp)***


---

|       |                                                                                                                                                                                                           |          |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| 説明    | ストリーム入出力用ファイルをクローズします。                                                                                                                                                                                    |          |
| ヘッダ   | <stdio.h>                                                                                                                                                                                                 |          |
| リターン値 | 正常：0<br>異常：0 以外                                                                                                                                                                                           |          |
| 引数    | fp                                                                                                                                                                                                        | ファイルポインタ |
| 例     | <pre>#include &lt;stdio.h&gt; FILE *fp; int ret; ret=fclose(fp);</pre>                                                                                                                                    |          |
| 備考    | <p>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。</p> <p>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。</p> |          |

***int fflush(FILE \*fp)***


---

|       |                                                                                                                                                   |          |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| 説明    | 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。                                                                                                                 |          |
| ヘッダ   | <stdio.h>                                                                                                                                         |          |
| リターン値 | 正常：0<br>異常：0 以外                                                                                                                                   |          |
| 引数    | fp                                                                                                                                                | ファイルポインタ |
| 例     | <pre>#include &lt;stdio.h&gt; FILE *fp; int ret; ret=fflush(fp);</pre>                                                                            |          |
| 備考    | <p>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。</p> |          |



## ファイルオープン

***FILE \*fopen(const char \*fname, const char \*mode)***

説明 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。

ヘッダ <stdio.h>

リターン値 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

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

```
例
#include <stdio.h>
FILE *ret;
const char *fname, *mode;
ret=fopen(fname,mode);
```

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。

## ファイル再オープン

***FILE \*freopen(const char \*fname, const char \*mode, FILE \*fp)***

説明 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。

ヘッダ <stdio.h>

リターン値 正常 : fp  
異常 : NULL

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

例  

```
#include <stdio.h>
const char *fname, *mode;
FILE *ret, *fp;
ret=freopen(fname,mode,fp);
```

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くて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。

## バッファ領域設定

***void setbuf(FILE \*fp, char buf[BUFSIZ])***

|     |                                                                                                                                                                   |              |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| 説明  | 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。                                                                                                                             |              |
| ヘッダ | <code>&lt;stdio.h&gt;</code>                                                                                                                                      |              |
| 引数  | <code>fp</code>                                                                                                                                                   | ファイルポインタ     |
|     | <code>buf</code>                                                                                                                                                  | バッファ領域へのポインタ |
| 例   | <pre>#include &lt;stdio.h&gt; FILE *fp; char buf[BUFSIZ];     setbuf(fp,buf);</pre>                                                                               |              |
| 備考  | setbuf 関数は、ファイルポインタ <code>fp</code> の示すストリーム入出力用ファイルに対して <code>buf</code> 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定義します。この結果、大きさが <code>BUFSIZ</code> のバッファ領域を使用した入出力処理が行われます。 |              |

***int setvbuf(FILE \*fp, char \*buf, int type, size\_t size)***

説明 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。

ヘッダ <stdio.h>

リターン値 正常：0  
異常：0 以外

|     |      |              |
|-----|------|--------------|
| 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   |

```
例
#include <stdio.h>
FILE *fp;
char *buf;
int type, ret;
size_t size;
ret=setvbuf(fp,buf,type,size);
```

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となります。
- (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。

## 書式付きファイル出力

***int fprintf(FILE \*fp, const char \*control [, arg ...])***

説明 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。

ヘッダ <stdio.h>

リターン値 正常：変換し出力した文字数  
異常：負の値

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

```
例
#include <stdio.h>
FILE *fp;
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
ret=fprintf(fp,control,buffer);
```

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】

書式を表わす文字列は、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。

$$\%[\text{フラグ}...] \left\{ \begin{array}{l} [ * ] \\ [ \text{フィールド幅} ] \end{array} \right\} \left( \begin{array}{l} [ * ] \\ [ \text{精度} ] \end{array} \right) [\text{パラメタのサイズ指定}] \text{変換文字}$$

この変換仕様に対して、実際に出力するパラメタが無い時は、その動作は保証されません。また、変換仕様よりも実際に出力するパラメタの個数が多い時は、余分なパラメタ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 10.33 に示します。

表 10.33 フラグの種類と意味

| 項目   | 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。                                                                                                                                                                                                                  |
| 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。<br>「+」と共に使用した時、本フラグは無視されます。                                                                                                                                                                                           |
| 4 #  | 表 10.35 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。<br>1. c, d, i, s, u 変換の時<br>本フラグは無視されます。<br>2. o 変換の時<br>変換したデータの先頭に 0 を付けます。<br>3. x(あるいは X)変換の時<br>変換したデータの先頭に 0x(あるいは 0X)を付けます。<br>4. e, E, f, g, G 変換の時<br>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 10.35 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) パラメタのサイズ指定

d, i, o, u, x, X, e, E, f, g, G 変換の時(表 10.35 参照)

変換するデータのサイズ(short 型、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 10.34 にサイズ指定の種類とその意味を示します。

表 10.34 パラメ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。        |
| 2 | 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。 |
| 3 | 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。                          |

## (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されません。表 10.35 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、その動作は保証されません。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 10.35 変換文字と変換の方式

|    | 変換文字 | 変換の種類  | 変換の方式                                                                                                                                       | 変換の対象とするデータの型 | 精度に対する注意事項                                                                                                                                    |
|----|------|--------|---------------------------------------------------------------------------------------------------------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| 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型データを符号無しの8進数の文字列に変換します。                                                                                                                 | int型          |                                                                                                                                               |
| 4  | u    | u変換    | int型データを符号無しの10進数の文字列に変換します。                                                                                                                | int型          |                                                                                                                                               |
| 5  | x    | x変換    | int型データを符号無しの16進数の文字列に変換します。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型データを符号無しの16進数の文字列に変換します。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.dd」の形式の10進数の文字列に変換します。                                                                                                   | double型       |                                                                                                                                               |
| 8  | 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わします。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。 |
| 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       |                                                                                                                                               |
| 10 | g    | g変換(ある | 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。                                                                                                                  |
| 11 | G    | いはG変換) |                                                                                                                                             | double型       |                                                                                                                                               |
| 12 | 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。                                                                                                                                   |



|    | 変換文字 | 変換の種類         | 変換の方式                                                                                                        | 変換の対象とするデータの型 | 精度に対する注意事項                                                                                                 |
|----|------|---------------|--------------------------------------------------------------------------------------------------------------|---------------|------------------------------------------------------------------------------------------------------------|
| 13 | 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ず(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)。 |
| 14 | 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。                                                                                                |
| 15 | 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  |                                                                                                            |
| 16 | %    | 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。

***int fscanf(FILE \*fp, const char \*control [, ptr ...])***

説明 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。

ヘッダ <stdio.h>

リターン値 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

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

例

```
#include <stdio.h>
FILE *fp;
const char *control="%d";
int ret,buffer[10];
ret=fscanf(fp,control,buffer);
```

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】

書式を表わす文字列は、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わ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、書式を表わ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、その動作は保証されません。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

入力したデータをパラメタ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 10.37 参照)、変換後のデータのサイズ(short 型、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 10.36 にサイズ指定の種類とその意味を示します。

表 10.36 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。<br>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。                                                      |

・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 10.37 変換文字と変換の内容

| 変換文字 | 変換の種類 | 変換の方式                 | 対応するパラメタのデータ型                                                                                                                                                                                                                                                                         |               |
|------|-------|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| 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                  | 符号無し 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。                                                                                                                                                                                                                                                              | 整数型           |
| 6    | X     | X 変換                  | x 変換 と X 変換 に意味の違いはありません。                                                                                                                                                                                                                                                             |               |
| 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、指定分の大きさが重要です。                                                                                                                       | char 型        |
| 9    | e     | e 変換                  | 浮動小数点数を示す文字列を浮動小数点型データに変換します。                                                                                                                                                                                                                                                         | 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<br>変換は生<br>じません。 | データの <input/> は行わず、いままでに <input/> したデータの文字数が設定されます。                                                                                                                                                                                                                                   | 整数型           |
| 16   | [     | [変換                   | [の後に文字の集合、その後]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、 <input/> 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 <input/> 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 <input/> します。 <input/> 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<br>変換は生<br>じません。 | %を読み込みます。                                                                                                                                                                                                                                                                             | なし            |

変換文字が表 10.37 に示す文字以外の英文字の時は、その動作は保証されません。また、その他の文字の時は、その動作は処理系定義です。

## 書式付き出力

***int printf(const char \*control [, arg ...])***

|       |                                                                                                                                   |                                  |
|-------|-----------------------------------------------------------------------------------------------------------------------------------|----------------------------------|
| 説明    | 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。                                                                                             |                                  |
| ヘッダ   | <stdio.h>                                                                                                                         |                                  |
| リターン値 | 正常：変換し出力した文字数<br>異常：負の値                                                                                                           |                                  |
| 引数    | control<br>arg, ...                                                                                                               | 書式を示す文字列へのポインタ<br>書式に従って出力されるデータ |
| 例     | <pre>#include &lt;stdio.h&gt; const char *control="%s"; int ret; char buffer[]="Hello World\n"; ret=printf(control,buffer);</pre> |                                  |
| 備考    | printf関数は、controlが指す書式を示す文字列に従って、パラメタ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。                               |                                  |

## 書式付き入力

***int scanf(const char \*control [, ptr ...])***

|       |                                                                                                                                                                                                                                                                 |                                           |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| 説明    | 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。                                                                                                                                                                                                                           |                                           |
| ヘッダ   | <stdio.h>                                                                                                                                                                                                                                                       |                                           |
| リターン値 | 正常：入力変換に成功したデータの個数<br>異常：EOF                                                                                                                                                                                                                                    |                                           |
| 引数    | control<br>ptr, ...                                                                                                                                                                                                                                             | 書式を示す文字列へのポインタ<br>入力変換したデータを格納する記憶域へのポインタ |
| 例     | <pre>#include &lt;stdio.h&gt; const char *control="%d"; int ret,buffer[10]; ret=scanf(control, buffer);</pre>                                                                                                                                                   |                                           |
| 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。<br>%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。 |                                           |

***int sprintf(char \*s, const char \*control [, arg ...])***

説明 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。

ヘッダ <stdio.h>

リターン値 変換した文字数

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

例

```
#include <stdio.h>
char *s;
const char *control="%s";
int ret;
char buffer[]="Hello World\n";
ret=sprintf(s,control,buffer);
```

備考 sprintf関数は、control が指す書式を示す文字列に従って、パラメタ 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。

***int sscanf(const char \*s, const char \*control [, ptr ...])***

説明 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。

ヘッダ <stdio.h>

リターン値 正常：入力変換に成功したデータの個数  
異常：EOF

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

例

```
#include <stdio.h>
const char *s, *control="%d";
int ret,buffer[10];
ret=sscanf(s,control,buffer);
```

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。

## 可変個パラメタファイル出力

***int* *vfprintf*(*FILE* \**fp*, *const char* \**control*, *va\_list* *arg*)**

|       |                                                                                                                                                                                                                                                                                                                                                                                           |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 説 明   | 可変個のパラメタ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。                                                                                                                                                                                                                                                                                                                                               |
| ヘッダ   | <stdio.h>                                                                                                                                                                                                                                                                                                                                                                                 |
| リターン値 | 正常：変換し出力した文字数<br>異常：負の値                                                                                                                                                                                                                                                                                                                                                                   |
| 引 数   | <i>fp</i> ファイルポインタ<br><i>control</i> 書式を示す文字列へのポインタ<br><i>arg</i> 引数リスト                                                                                                                                                                                                                                                                                                                   |
| 例     | <pre>#include &lt;stdarg.h&gt; #include &lt;stdio.h&gt; FILE *fp; const char *control="%d"; int ret;  void prlist(int count ,...) {     va_list ap;     int i;     va_start(ap, count);     for(i=0;i&lt;count;i++)         ret=vfprintf(fp, control, ap);     va_end(ap); }</pre>                                                                                                        |
| 備 考   | <p><i>vfprintf</i> 関数は、<i>control</i> 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、<i>fp</i> の示すストリーム入出力用ファイルへ出力します。</p> <p><i>vfprintf</i> 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。</p> <p>また、<i>vfprintf</i> 関数では <i>va_end</i> マクロは呼び出しません。</p> <p>書式の仕様の詳細は <i>fprintf</i> 関数を参照してください。</p> <p>引数リストを示す <i>arg</i> は、<i>va_start</i> (およびそれに続く <i>va_arg</i> マクロ) によって初期化されていなければなりません。</p> |

***int vprintf(const char \*control, va\_list arg)***

説明 可変個のパラメタ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。

ヘッダ <stdio.h>

リターン値 正常：変換し出力した文字数  
異常：負の値

引数 control 書式を示す文字列へのポインタ  
arg 引数リスト

例

```
#include <stdarg.h>
#include <stdio.h>
FILE *fp;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
 va_list ap;
 int i;
 va_start(ap, count);
 for(i=0;i<count;i++)
 ret=vprintf(control, ap);
 va_end(ap);
}
```

備考 vprintf 関数は、control が指す書式を示す文字列に従って、可変個のパラメタリストを順に変換、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。



## 可変個パラメタ文字列出力

***int vsprintf(char \*s, const char \*control, va\_list arg)***

説明 可変個のパラメタリストを書式に従って、指定した記憶域に出力します。

ヘッダ <stdio.h>

リターン値 正常：変換した文字数  
異常：負の数

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

例

```
#include <stdarg.h>
#include <stdio.h>
char *s;
const char *control="%d";
int ret;

void prlist(int count ,...)
{
 va_list ap;
 int i;
 va_start(ap, count);
 for(i=0;i<count;i++)
 ret=vsprintf(s,control,buffer);
 va_end(ap);
}
```

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附加されます。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。

***int fgetc(FILE \*fp)***

- 説明** ストリーム入出力用ファイルから1文字入力します。
- ヘッダ** <stdio.h>
- リターン値** 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
- 引数** fp                      ファイルポインタ
- 例**
- ```
#include <stdio.h>
FILE *fp;
int ret;
ret=fgetc(fp);
```
- エラー条件** 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。
- 備考** 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際は、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。

ファイルから文字列入力

char *fgets(char *s, int n, FILE *fp)

説明 ストリーム入出力用ファイルから文字列を入力します。

ヘッダ <stdio.h>

リターン値 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

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

例

```
#include <stdio.h>
char *s, *ret;
int n;
FILE *fp;
ret=fgets(s,n,fp);
```

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際は 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、エラー発生の際は、s が指す記憶域の内容は保証されません。

int fputc(int c, FILE *fp)

説明 ストリーム入出力用ファイルへ1文字出力します。

ヘッダ <stdio.h>

リターン値 正常：出力した文字
異常：EOF

引 数 c 出力する文字
 fp ファイルポインタ

例 #include <stdio.h>
 FILE *fp;
 int c, ret;
 ret=fputc(c,fp);

エラー条件 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。

備 考 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、エラー発生の際は、EOF を返します。

ファイルに文字列出力

int fputs(const char *s, FILE *fp)

説明 ストリーム入出力用ファイルへ文字列を出力します。

ヘッダ <stdio.h>

リターン値 正常: 0
異常: 0 以外

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

例

```
#include <stdio.h>
const char *s;
int ret;
FILE *fp;
ret=fputs(s,fp);
```

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、エラー発生の際は、0 以外の値を返します。

int getc(FILE *fp)

説明	ストリーム入出力用ファイルから1文字入力します。
ヘッダ	<stdio.h>
リターン値	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
引数	fp ファイルポインタ
例	<pre>#include <stdio.h> FILE *fp; int ret; ret=getc(fp);</pre>
エラー条件	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。
備考	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際は、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。

int getchar(void)

説明	標準入力ファイル(stdin)から、1文字入力します。
ヘッダ	<stdio.h>
リターン値	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
例	<pre>#include <stdio.h> int ret; ret=getchar();</pre>
エラー条件	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。
備考	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際は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。

文字列入力

char *gets(char *s)

説明	標準入力ファイル(stdin)から文字列を入力します。
ヘッダ	<stdio.h>
リターン値	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
引数	s 文字列を入力する記憶域へのポインタ
例	<pre>#include <stdio.h> char *ret, *s; ret=gets(s);</pre>
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際は、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、エラー発生の際は s が指す記憶域の内容は保証されません。

ファイルに1文字出力

int putc(int c, FILE *fp)

説明	ストリーム入出力用ファイルへ1文字出力します。
ヘッダ	<stdio.h>
リターン値	正常：出力した文字 異常：EOF
引数	c 出力する文字 fp ファイルポインタ
例	<pre>#include <stdio.h> FILE *fp; int c, ret; ret=putc(c, fp);</pre>
エラー条件	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。
備考	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際は、EOF を返します。

int putchar(int c)

説明	標準出力ファイル(stdout)へ1文字出力します。
ヘッダ	<stdio.h>
リターン値	正常：出力した文字 異常：EOF
引数	c 出力する文字
例	<pre>#include <stdio.h> int c, ret; ret=putchar(c);</pre>
エラー条件	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。
備考	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、エラー発生の際はEOFを返します。

int puts(const char *s)

説明	標準出力ファイル(stdout)へ文字列を出力します。
ヘッダ	<stdio.h>
リターン値	正常：0 異常：0以外
引数	s 出力する文字列へのポインタ
例	<pre>#include <stdio.h> const char *s; int ret; ret=puts(s);</pre>
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、0以外の値を返します。

ファイルに1文字返却

int ungetc(int c, FILE *fp)

説明 ストリーム入出力用ファイルへ1文字を戻します。

ヘッダ <stdio.h>

リターン値 正常：戻した文字
異常：EOF

引数 c 戻す文字
fp ファイルポインタ

```
例
#include <stdio.h>
int c, ret;
FILE *fp;
ret=ungetc(c, fp);
```

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、エラー発生の際は 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されません。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、位置指示子は保証されません。

size_t fread(void *ptr, size_t size, size_t n, FILE *fp)

説明 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。

ヘッダ <stdio.h>

リターン値 `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

引数	<code>ptr</code>	データを入力する記憶域へのポインタ
	<code>size</code>	1 メンバのバイト数
	<code>n</code>	入力するメンバの数
	<code>fp</code>	ファイルポインタ

例

```
#include <stdio.h>
void *ptr;
size_t size;
size_t n, ret;
FILE *fp;
    ret=fread(ptr,size,n,fp);
```

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際は、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されません。

ファイル書き込み

size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp)

説明 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。

ヘッダ <stdio.h>

リターン値 出力に成功したメンバ数

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

例

```
#include <stdio.h>
const void *ptr;
size_t size;
size_t n, ret;
FILE *fp;
    ret=fwrite(ptr,size,n,fp);
```

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、エラー発生の際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されません。

int fseek(FILE *fp, long offset, int type)

説明 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。

ヘッダ <stdio.h>

リターン値 正常：0
異常：0 以外

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

例

```
#include <stdio.h>
FILE *fp;
long offset;
int type, ret;
ret=fseek(fp,offset,type);
```

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 10.38 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 10.38 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。

ファイル読み書き位置取得

long ftell(FILE *fp)

説明	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。
ヘッダ	<stdio.h>
リターン値	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)
引数	fp ファイルポインタ
例	<pre>#include <stdio.h> FILE *fp; long ret; ret=ftell(fp);</pre>
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 関数で使え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わすことにはなりません。

ファイル先頭に移動

void rewind(FILE *fp)

説明	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。
ヘッダ	<stdio.h>
引数	fp ファイルポインタ
例	<pre>#include <stdio.h> FILE *fp; rewind(fp);</pre>
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。

void clearerr(FILE *fp)

説明	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。
ヘッダ	<stdio.h>
引数	fp ファイルポインタ
例	<pre>#include <stdio.h> FILE *fp; clearerr(fp);</pre>
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。

int feof(FILE *fp)

説明	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。
ヘッダ	<stdio.h>
リターン値	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
引数	fp ファイルポインタ
例	<pre>#include <stdio.h> FILE *fp; int ret; ret=feof(fp);</pre>
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。

ファイルエラー状態判定

int ferror(FILE *fp)

説明	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。
ヘッダ	<stdio.h>
リターン値	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
引数	fp ファイルポインタ
例	<pre>#include <stdio.h> FILE *fp; int ret; ret=ferror(fp);</pre>
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。

エラーメッセージ出力

void perror(const char *s)

説明	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。
ヘッダ	<stdio.h>
引数	s エラーメッセージへのポインタ
例	<pre>#include <stdio.h> const char *s; perror(s);</pre>
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んと空白とその後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。

(13) <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	div_t	div 関数のリターン値の構造体の型です。
(マクロ)	ldiv_t	ldiv 関数のリターン値の構造体の型です。
定数	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。
(マクロ)		
関数	atof	数を表現する文字列を double 型の浮動小数点数値に変換します。
	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。
	strtod	数を表現する文字列を double 型の浮動小数点数値に変換します。
	strtol	数を表現する文字列を long 型の整数値に変換します。
	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、すべての割当てられた記憶域を 0 によって初期化します。
	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。

文字列を *double* 型に変換***double atof(const char *nptr)***

説明	数を表現する文字列を、 <i>double</i> 型の浮動小数点数値に変換します。
ヘッダ	<stdlib.h>
リターン値	変換された <i>double</i> 型の浮動小数点数値
引数	<i>nptr</i> 変換する数を表現する文字列のポインタ
例	<pre>#include <stdlib.h> const char *nptr; double ret; ret=atof(nptr);</pre>
備考	変換は、浮動小数点数の形式に合わない最初の文字までに対して行います。 <i>atof</i> 関数は、オーバーフロー等のエラーが生じても <i>errno</i> を設定しません。また、エラーが生じた場合、結果の値は保証されません。変換のエラーが生じる可能性がある場合は、 <i>strtod</i> 関数を使用してください。

文字列を *int* 型に変換***int atoi(const char *nptr)***

説明	10進数を表現する文字列を、 <i>int</i> 型の整数値に変換します。
ヘッダ	<stdlib.h>
リターン値	変換された <i>int</i> 型の整数値
引数	<i>nptr</i> 変換する数を表現する文字列のポインタ
例	<pre>#include <stdlib.h> const char *nptr; int ret; ret=atoi(nptr);</pre>
備考	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。 <i>atoi</i> 関数は、オーバーフロー等のエラーが生じても <i>errno</i> を設定しません。また、エラーが生じた場合、結果の値を保証しません。変換のエラーが生じる可能性がある場合は、 <i>strtol</i> 関数を使用してください。

long atol(const char *nptr)

説 明 10 進数を表現する文字列を、long 型の整数値に変換します。

ヘッダ <stdlib.h>

リターン値 変換された long 型の整数値

引 数 nptr 変換する数を表現する文字列のポインタ

例

```
#include <stdlib.h>
const char *nptr;
long ret;
ret=atol(nptr);
```

備 考 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、オーバーフロー等のエラーが生じても `errno` を設定しません。また、エラーが生じた場合、結果の値を保証しません。変換時にエラーが生じる可能性がある場合は、`strtol` 関数を使用してください。

文字列を *double* 型に変換***double strtod(const char *nptr, char **endptr)***

説明	数を表現する文字列を <i>double</i> 型の浮動小数点数値に変換します。	
ヘッダ	<stdlib.h>	
リターン値	正常 :	<i>nptr</i> が指している文字列が浮動小数点数を構成しない文字で始まっている時 : 0 <i>nptr</i> が指している文字列が浮動小数点数を構成する文字で始まっている時 : 変換された <i>double</i> 型の浮動小数点数値
	異常 :	変換後の値がオーバーフローの時 : 変換する文字列の符号と同符号をもつ <i>HUGE_VAL</i> 変換後の値がアンダフローの時 : 0
引数	<i>nptr</i> <i>endptr</i>	変換する数を表現する文字列へのポインタ 浮動小数点数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
例	<pre>#include <stdlib.h> const char *nptr; char **endptr; double ret; ret=strtod(nptr, endptr);</pre>	
エラー条件	変換後の値がオーバーフロー / アンダフローをおこした時は <i>errno</i> に <i>ERANGE</i> が設定されます。	
備考	<i>strtod</i> 関数は、最初の数字もしくは小数点から浮動小数点数値を構成しない文字の直前までを「10.1.3(4) 浮動小数点演算の仕様」の規則に従って <i>double</i> 型の浮動小数点数値に変換します。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して仮定されます。 <i>endptr</i> の指す領域には、浮動小数点数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数を構成しない文字がある場合は <i>nptr</i> の値を設定します。 <i>endptr</i> が <i>NULL</i> の場合、この設定は行われません。	

long strtol(const char *nptr, char **endptr, int base)

説明	数を表現する文字列を long 型の整数値に変換します。
ヘッダ	<stdlib.h>
リターン値	<p>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</p> <p>異常： 変換後の値がオーバーフローの時：変換する文字列の符号に従って LONG_MAX あるいは LONG_MIN</p>
引 数	<p>nptr 変換する数を表現する文字列へのポインタ</p> <p>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</p> <p>base 変換の基数(0 又は 2~36)</p>
例	<pre>#include <stdlib.h> long ret; const char *nptr; char **endptr; int base; ret=strtol(nptr,endptr,base);</pre>
エラー条件	変換後の値がオーバーフローをおこした時は、errno に ERANGE が設定されます。
備 考	<p>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。</p> <p>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。</p> <p>base の値が 0 の時は、「10.1.1(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。</p>

擬似乱数生成

int rand(void)

- 説明 0 から RAND_MAX の間の擬似乱数整数を生成します。
- ヘッダ <stdlib.h>
- リターン値 擬似乱数整数値
- 例
- ```
#include <stdlib.h>
int ret;
ret=rand();
```

## 擬似乱数列初期設定

***void srand(unsigned int seed)***

- 説明 rand 関数で生成する擬似乱数列の初期値を設定します。
- ヘッダ <stdlib.h>
- 引数 seed 擬似乱数列生成の初期値
- 例
- ```
#include <stdlib.h>
unsigned int seed;
srand(seed);
```
- 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。

初期化付き記憶域割り当て

void *calloc(size_t nelem, size_t elsize)

説明	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 によって初期化します。	
ヘッダ	<stdlib.h>	
リターン値	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、またはパラメタのいずれかが 0 の時：NULL	
引数	nelem	要素の数
	elsize	一つの要素の占めるバイト数
例	<pre>#include <stdlib.h> size_t nelem, elsize; void *ret; ret=calloc(nelem,elsize);</pre>	
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。	

記憶域解放

void free(void *ptr)

説明	指定された記憶域を解放します。	
ヘッダ	<stdlib.h>	
引数	ptr	解放する記憶域のアドレス
例	<pre>#include <stdlib.h> void *ptr; free(ptr);</pre>	
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されません。また、解放された後の記憶域を参照した時の動作も保証されません。	

記憶域割り当て

void *malloc(size_t size)

説明	記憶域を割り当てます。	
ヘッダ	<stdlib.h>	
リターン値	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	
引数	size	割り当てる記憶域のバイト数
例	<pre>#include <stdlib.h> size_t size; void *ret; ret=malloc(size);</pre>	
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。	

記憶域割り当てサイズ変更

void *realloc(void *ptr, size_t size)

説明	記憶域の大きさを指定された大きさに変更します。	
ヘッダ	<stdlib.h>	
リターン値	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	
引数	ptr size	変更する記憶域の先頭アドレス 変更後の記憶域のバイト数
例	<pre>#include <stdlib.h> size_t size; void *ptr, *ret; ret=realloc(ptr,size);</pre>	
備考	<p>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。</p> <p>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、動作は保証されません。</p>	

```
void *bsearch(const void *key, const void *base, size_t nmemb, size_t size,
              int (*compar)(const void *, const void *))
```

説明 二分探索を行います。

ヘッダ <stdlib.h>

リターン値 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

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

例

```
#include <stdlib.h>
const void *key, *base;
size_t nmemb, size;
int (*compar)(const void *, const void *);
void *ret;
    ret=bsearch(key,base,nmemb,size,compar);
```

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探索法によって検索します。比較を行う関数は、比較する二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。

ソート

```
void qsort(const void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *))
```

説明 ソートを行います。

ヘッダ <stdlib.h>

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

例

```
#include <stdlib.h>
const void *base;
size_t nmemb, size;
int (*compar)(const void *, const void *);
qsort(base, nmemb, size, compar);
```

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、比較する二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。

絶対値

```
int abs(int i)
```

説明 int型整数の絶対値を求めます。

ヘッダ <stdlib.h>

リターン値 iの絶対値

引数 i 絶対値を求める整数

例

```
#include <stdlib.h>
int i, ret;
ret=abs(i);
```

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されません。

div_t div(int numer, int denom)

説明	int 型整数の除算の商と余りを計算します。	
ヘッダ	<stdlib.h>	
リターン値	numer を denom で除算した結果の商と余り	
引数	numer	被除数
	denom	除数
例	<pre>#include <stdlib.h> int numer, denom; div_t ret; ret=div(numer,denom);</pre>	

long labs(long j)

説明	long 型整数の絶対値を計算します。	
ヘッダ	<stdlib.h>	
リターン値	j の絶対値	
引数	j	絶対値を求める整数
例	<pre>#include <stdlib.h> long j; long ret; ret=labs(j);</pre>	
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されません。	

ldiv_t ldiv(long numer, long denom)

説明 long 型整数の除算の商と余りを計算します。

ヘッダ <stdlib.h>

リターン値 numer を denom で除算した結果の商と余り

引数	numer	被除数
	denom	除数

例

```
#include <stdlib.h>
long numer, denom;
ldiv_t ret;
ret=ldiv(numer,denom);
```

(14) <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	指定された二つの記憶域の比較を行います。
	strcmp	指定された二つの文字列を比較します。
	strncmp	指定された二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	文字列の長さを計算します。
	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	「12.3 標準ライブラリのエラーメッセージ」を参照してください。

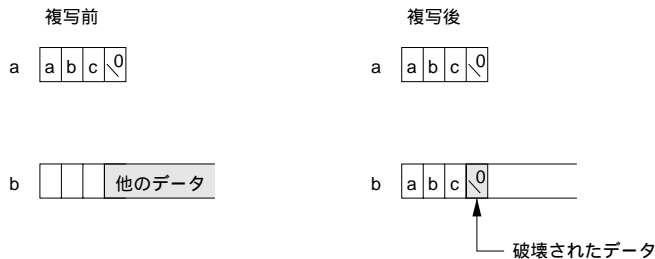
本標準インクルードファイル内で定義されている関数を使用する時は、以下の二つの事項に注意する必要があります。

- (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、動作は保証され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。

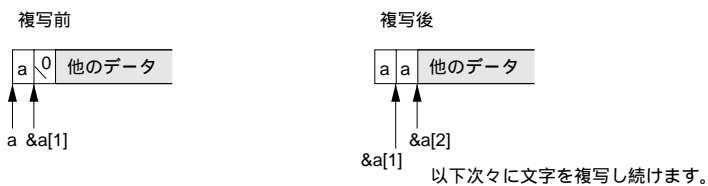


- (2) 文字列の複写を行う時、複写元の領域と複写先の領域が重なっていると正しい動作が保証されませんので注意が必要です。

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。



void *memcpy(void *s1, const void *s2, size_t n)

説明 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。

ヘッダ <string.h>

リターン値 s1 の値

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

例

```
#include <string.h>
void *ret, *s1;
const void *s2;
size_t n;
ret=memcpy(s1,s2,n);
```

char *strcpy(char *s1, const char *s2)

説明 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。

ヘッダ <string.h>

リターン値 s1 の値

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

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
ret=strcpy(s1,s2);
```

文字列複写

char *strncpy(char *s1, const char *s2, size_t n)

説明 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。

ヘッダ <string.h>

リターン値 s1 の値

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

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
size_t n;
ret=strncpy(s1,s2,n);
```

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長さが 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長さが 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。

文字列連結

char *strcat(char *s1, const char *s2)

説明 文字列の後に、文字列を連結します。

ヘッダ <string.h>

リターン値 s1 の値

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

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
ret=strcat(s1,s2);
```

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。

char *strncat(char *s1, const char *s2, size_t n)

説明 文字列に文字列を指定した文字数分連結します。

ヘッダ <string.h>

リターン値 s1 の値

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

例

```
#include <string.h>
char *s1, *ret;
const char *s2;
size_t n;
ret=strncat(s1,s2,n);
```

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。

int memcmp(const void *s1, const void *s2, size_t n)

説明 指定された二つの記憶域の内容を比較します。

ヘッダ <string.h>

リターン値 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

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

例

```
#include <string.h>
const void *s1, *s2;
size_t n;
int ret;
ret=memcmp(s1,s2,n);
```

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。

文字列比較

int strcmp(const char *s1, const char *s2)

説明 指定された二つの文字列の内容を比較します。

ヘッダ <string.h>

リターン値 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

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

例

```
#include <string.h>
const char *s1, *s2;
int ret;
ret=strcmp(s1,s2);
```

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。

文字列比較

int strncmp(const char *s1, const char *s2, size_t n)

説明 指定された二つの文字列を指定された文字分まで比較します。

ヘッダ <string.h>

リターン値 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

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

例

```
#include <string.h>
const char *s1, *s2;
size_t n;
int ret;
ret=strncmp(s1,s2,n);
```

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。

void *memchr(const void *s, int c, size_t n)

説明 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。

ヘッダ <string.h>

リターン値 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

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

例

```
#include <string.h>
const void *s;
int c;
size_t n;
void *ret;
ret=memchr(s,c,n);
```

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。

char *strchr(const char *s, int c)

説明 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。

ヘッダ <string.h>

リターン値 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

引数 s 検索を行う文字列へのポインタ
 c 検索する文字

例

```
#include <string.h>
const char *s;
int c;
char *ret;
ret=strchr(s,c);
```

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。

指定文字群が最初に現れるまでの文字数***size_t strcspn(const char *s1, const char *s2)***

説明	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。
ヘッダ	<string.h>
リターン値	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
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
例	<pre>#include <string.h> const char *s1, *s2; size_t ret; ret=strcspn(s1,s2);</pre>
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、その文字列の長さ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わすヌル文字は、s2 で指された文字列の一部とはみなされません。

指定文字群が最初に現れる位置***char *strpbrk(const char *s1, const char *s2)***

説明	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。
ヘッダ	<string.h>
リターン値	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
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
例	<pre>#include <string.h> const char *s1, *s2; char *ret; ret=strpbrk(s1,s2);</pre>
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。

char *strrchr(const char *s, int c)

説明	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。	
ヘッダ	<string.h>	
リターン値	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
引数	s	検索を行う文字列へのポインタ
	c	検索する文字
例	<pre>#include <string.h> const char *s; int c; char *ret; ret=strrchr(s,c);</pre>	
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わすヌル文字も検索の対象として含まれます。	

size_t strspn(const char *s1, const char *s2)

説明	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。	
ヘッダ	<string.h>	
リターン値	s1 の先頭から、s2 で指定した文字が続いている文字数	
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
例	<pre>#include <string.h> const char *s1, *s2; size_t ret; ret=strspn(s1,s2);</pre>	
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、その文字列の長さをリターン値として返します。	

最初の文字列位置***char *strstr(const char *s1, const char *s2)***

説明 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。

ヘッダ <string.h>

リターン値 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

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

例

```
#include <string.h>
const char *s1, *s2;
char *ret;
ret=strstr(s1,s2);
```

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。

char *strtok(char *s1, const char *s2)

説明 指定した文字列をいくつかの字句に切り分けます。

ヘッダ <string.h>

リターン値 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

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

例
#include <string.h>
char *s1, *ret;
const char *s2;
ret=strtok(s1,s2);

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 パラメタには 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。

文字繰り返し

void *memset(void *s, int c, size_t n)

説明 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。

ヘッダ <string.h>

リターン値 s の値

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

例

```
#include <string.h>
void *s, *ret;
int c;
size_t n;
ret=memset(s,c,n);
```

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。

エラーメッセージ文字列

char *strerror(int s)

説明 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。

ヘッダ <string.h>

リターン値 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

引数	s	エラー番号
----	---	-------

例

```
#include <string.h>
char *ret;
int s;
ret=strerror(s);
```

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、動作は保証されません。

size_t strlen(const char *s)

説明	文字列の長さを計算します。	
ヘッダ	<string.h>	
リターン値	文字列の文字数	
引数	s	長さを求める文字列へのポインタ
例	<pre>#include <string.h> const char *s; size_t ret; ret=strlen(s);</pre>	
備考	s が指す文字列の終了を表わすヌル文字は、文字列の長さとしては計算に入れません。	

void *memmove(void *s1, const void *s2, size_t n)

説明	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。	
ヘッダ	<string.h>	
リターン値	s1 の値	
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
例	<pre>#include <string.h> void *ret, *s1; const void *s2; size_t n; ret=memmove(s1,s2,n);</pre>	

10.3.2 EC++クラスライブラリ

(1) ライブラリの概要

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 10.39 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 10.39 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

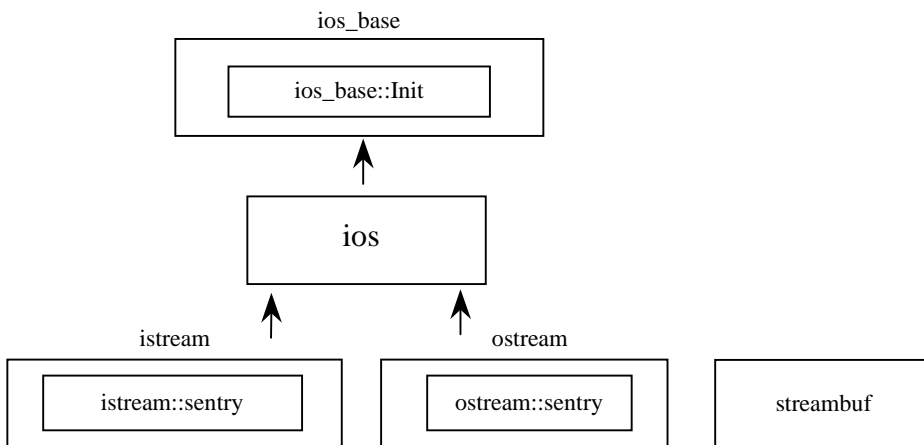
ライブラリの種類	内容	標準インクルードファイル
1 ストリーム入出力用クラスライブラリ	入出力操作を行うライブラリです。	<ios>, <streambuf>, <iostream>, <ostream>, <iostream>, <iomanip>
2 メモリ操作用ライブラリ	メモリの確保・解放を行うライブラリです。	<new>
3 複素数計算用クラスライブラリ	複素数データ演算を行うライブラリです。	<complex>
4 文字列操作用クラスライブラリ	文字列操作を行うライブラリです。	<string>

(2) 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() ~Init()	コンストラクタです 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。

10. C/C++言語仕様

(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。
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(mask&fmtflg 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開きます。
const ios_base::openmode ios_base::out         = 0x02; 出力用のファイルを開きます。
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)	初期設定を行います。
	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。
	iosstate rdstate() const	状態フラグ(state)を参照します。
	void clear(iosstate 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(iosstate 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

`iosstate 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(iosstate 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(iosstate 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 クラスマネピュレータ

種別	定義名	説明
関数	<code>ios_base& boolalpha(ios_base& str)</code>	bool 型の書式に設定します。
	<code>ios_base& noboolalpha(ios_base& str)</code>	bool 型の書式をクリアします。
	<code>ios_base& showbase(ios_base& str)</code>	基数表示接頭辞モードに設定します。
	<code>ios_base& noshowbase(ios_base& str)</code>	基数表示接頭辞モードをクリアします。
	<code>ios_base& showpoint(ios_base& str)</code>	小数点生成モードに設定します。
	<code>ios_base& noshowpoint(ios_base& str)</code>	小数点生成モードをクリアします。
	<code>ios_base& showpos(ios_base& str)</code>	+記号生成モードに設定します。
	<code>ios_base& noshowpos(ios_base& str)</code>	+記号生成モードをクリアします。
	<code>ios_base& skipws(ios_base& str)</code>	空白読み飛ばしモードに設定します。
	<code>ios_base& noskipws(ios_base& str)</code>	空白読み飛ばしモードをクリアします。
	<code>ios_base& uppercase(ios_base& str)</code>	大文字変換モードに設定します。
	<code>ios_base& nouppercase(ios_base& str)</code>	大文字変換モードをクリアします。
	<code>ios_base& internal(ios_base& str)</code>	内部補充モードに設定します。
	<code>ios_base& left(ios_base& str)</code>	左側補充モードに設定します。
	<code>ios_base& right(ios_base& str)</code>	右側補充モードに設定します。
	<code>ios_base& dec(ios_base& str)</code>	10 進モードに設定します。
	<code>ios_base& hex(ios_base& str)</code>	16 進モードに設定します。
	<code>ios_base& oct(ios_base& str)</code>	8 進モードに設定します。
	<code>ios_base& fixed(ios_base& str)</code>	固定小数点モードに設定します。
	<code>ios_base& scientific(ios_base& str)</code>	科学表記法モードに設定します。

`ios_base& boolalpha(ios_base& str)`

bool 型の書式に設定します。
リターン値は str です。

`ios_base& noboolalpha(ios_base& str)`

bool 型の書式をクリアします。
リターン値は str です。

`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。
	関数	char* _ec2p_getflag() const
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。

種別	定義名	説明
関数	<code>int sgetn(char* s, streamsize n)</code>	sの指す記憶領域にn個の文字を設定します。
	<code>int_type sputbackc(char c)</code>	読み込み位置をブットバックします。
	<code>int sungetc()</code>	読み込み位置をブットバックします。
	<code>int sputc(char c)</code>	文字cを挿入します。
	<code>int_type sputn(const char* s, streamsize n)</code>	sの指すn個の文字を挿入します。
	<code>char* eback() const</code>	入力ストリームの先頭ポインタを求めます。
	<code>char* gptr() const</code>	入力ストリームの次ポインタを求めます。
	<code>char* egptr() const</code>	入力ストリームの最後尾ポインタを求めます。
	<code>void gbump(int n)</code>	入力ストリームの次ポインタをn進めます。
	<code>void setg(char* gbeg, char* gnext, char* gend)</code>	入力ストリームの各ポインタを代入します。
	<code>char* pbase() const</code>	出力ストリームの先頭ポインタを求めます。
	<code>char* pptr() const</code>	出力ストリームの次ポインタを求めます。
	<code>char* epptr() const</code>	出力ストリームの最後尾ポインタを求めます。
	<code>void pbump(int n)</code>	出力ストリームの次ポインタをn進めます。
	<code>void setp(char* pbeg, char* pend)</code>	出力ストリームの各ポインタを設定します。
	<code>virtual streambuf* setbuf(char* s, streamsize n)¹</code>	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。
	<code>virtual pos_type seekoff(off_type off, ios_base::seekdir way, ios_base::openmode = (ios_base::openmode) (ios_base::in ios_base::out))¹</code>	ストリーム位置を変更します。
	<code>virtual pos_type seekpos(pos_type sp, ios_base::openmode = (ios_base::openmode) (ios_base::in ios_base::out))¹</code>	ストリーム位置を変更します。
	<code>virtual int sync()¹</code>	出力ストリームをフラッシュします。
	<code>virtual int showmanyc()¹</code>	入力ストリームの有効な文字数を求めます。
	<code>virtual streamsize xsgetn(char* s, streamsize n)</code>	sの指す記憶領域にn個の文字を設定します。
	<code>virtual int_type underflow()¹</code>	ストリーム位置を動かさずに一文字読み込みます。
	<code>virtual int_type uflow()¹</code>	次ポインタの一文字を読み込みます。
	<code>virtual int_type pbackfail(int_type c = eof)¹</code>	cによって示される文字をブットバックします。
	<code>virtual streamsize xsputn(const char* s, streamsize n)</code>	sの指すn個の文字を挿入します。
	<code>virtual int_type overflow(int_type c = eof)¹</code>	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。

`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、`setbuf(s,n)` 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

読み込み位置が有効でない場合 : 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ブットバックします。

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	<code>ok_</code>	入力可能状態か否かを意味します。
関数	<code>sentry(istream& is, bool noskipws = false)</code>	コンストラクタです。
	<code>~sentry()</code>	デストラクタです。
	<code>operator bool()</code>	<code>ok_</code> 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>>(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()	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)	
	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)	

10. C/C++言語仕様

種別	定義名	説明
関数	<code>istream& getline(char* s, streamsize n, char delim)</code>	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。
	<code>istream& getline(signed char* s, streamsize n, char delim)</code>	
	<code>istream& getline(unsigned char* s, streamsize n, char delim)</code>	
	<code>istream& ignore(streamsize n = 1, int_type delim = streambuf::eof)</code>	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。
	<code>int_type peek()</code>	次の入手可能な入力文字を求めます。
	<code>istream& read(char* s, streamsize n)</code>	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。
	<code>istream& read(signed char* s, streamsize n)</code>	
	<code>istream& read(unsigned char* s, streamsize n)</code>	
	<code>streamsize readsome(char* s, streamsize n)</code>	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。
	<code>streamsize readsome(signed char* s, streamsize n)</code>	
	<code>streamsize readsome(unsigned char* s, streamsize n)</code>	
	<code>istream& putback(char c)</code>	文字を入力ストリームに戻します。
	<code>istream& unget()</code>	入力ストリームの位置に戻します。
	<code>int sync()</code>	入力ストリームがあるかどうかを調べます。この関数は <code>streambuf::pubsync()</code> を呼び出します。
	<code>pos_type tellg()</code>	入力ストリームの位置を調べます。この関数は <code>streambuf::pubseekoff(0, cur, in)</code> を呼び出します。
<code>istream& seekg(pos_type pos)</code>	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 <code>streambuf::pubseekpos(pos)</code> を呼び出します。	
<code>istream& seekg(off_type off, ios_base::seekdir dir)</code>	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 <code>streambuf::pubseekoff(off, dir)</code> 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)

istream& istream::operator>>(short& n)

istream& istream::operator>>(unsigned short& n)

istream& istream::operator>>(int& n)

istream& istream::operator>>(unsigned int& n)

istream& istream::operator>>(long& n)

istream& istream::operator>>(unsigned long& n)

istream& istream::operator>>(float& n)

istream& istream::operator>>(double& n)

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(failbit)を呼び出して、streambuf::eof

istream& istream::get(char& c)

istream& istream::get(signed char& c)

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)

istream& istream::get(signed char* s, streamsize n)

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)`

`istream& istream::get(signed char* s, streamsize n, char delim)`

`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)`

`istream& istream::getline(signed char* s, streamsize n)`

`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)`

`istream& istream::getline(signed char* s, streamsize n, char delim)`

`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 : `rddbuf()->sgetc()`

`istream& istream::read(char* s, streamsize n)`

`istream& istream::read(signed char* s, streamsize n)`

`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)`

`streamsize istream::readsome(signed char* s, streamsize n)`

`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)	空白文字を読み飛ばします。

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	<code>istream& operator>>(istream& in, char* s)</code>	文字列を抽出し、sの指す記憶領域に格納します。
	<code>istream& operator>>(istream& in, signed char* s)</code>	
	<code>istream& operator>>(istream& in, unsigned char* s)</code>	
	<code>istream& operator>>(istream& in, char& c)</code>	文字を抽出し、cに格納します。
	<code>istream& operator>>(istream& in, signed char& c)</code>	
	<code>istream& operator>>(istream& in, unsigned char& c)</code>	

`istream& operator>>(istream& in, char* s)`

`istream& operator>>(istream& in, signed char* s)`

`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)`

`istream& operator>>(istream& in, signed char& c)`

`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_	出力可能状態が否かを意味します。
	__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。

10. C/C++言語仕様

(k) ostream クラス

種別	定義名	説明
関数	<code>ostream(streambuf* sbptr)</code>	コンストラクタです。
	<code>virtual ~ostream()</code>	デストラクタです。
	<code>ostream& operator<<(bool n)</code>	n を出力ストリームに挿入します。
	<code>ostream& operator<<(short n)</code>	
	<code>ostream& operator<<(unsigned short n)</code>	
	<code>ostream& operator<<(int n)</code>	
	<code>ostream& operator<<(unsigned int n)</code>	
	<code>ostream& operator<<(long n)</code>	
	<code>ostream& operator<<(unsigned long n)</code>	
	<code>ostream& operator<<(float n)</code>	
	<code>ostream& operator<<(double n)</code>	
	<code>ostream& operator<<(long double n)</code>	
	<code>ostream& operator<<(void* n)</code>	
	<code>ostream& operator<<(streambuf* sbptr)</code>	<code>sbptr</code> の出力列を出力ストリームに挿入します。
	<code>ostream& put(char c)</code>	文字 c を出力ストリームに挿入します。
	<code>ostream& write(const char* s, streamsize n)</code>	s の n 個の文字を出力ストリームに挿入します。
	<code>ostream& write(const signed char* s, streamsize n)</code>	
	<code>ostream& write(const unsigned char* s, streamsize n)</code>	
	<code>ostream& flush()</code>	出力ストリームをフラッシュします。この関数は <code>streambuf::pubsync()</code> を呼び出します。
	<code>pos_type tellp()</code>	現在の書き込み位置を求めます。この関数は <code>streambuf::pubseekoff(0,cur,out)</code> を呼び出します。
	<code>ostream& seekp(pos_type pos)</code>	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 <code>streambuf::pubseekpos(pos)</code> を呼び出します。
	<code>ostream& seekp(off_type off, seekdir dir)</code>	<code>dir</code> 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 <code>streambuf::pubseekoff(off,dir)</code> 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)`

`ostream& ostream::operator<<(short n)`

`ostream& ostream::operator<<(unsigned short n)`

`ostream& ostream::operator<<(int n)`

`ostream& ostream::operator<<(unsigned int n)`

`ostream& ostream::operator<<(long n)`

`ostream& ostream::operator<<(unsigned long n)`

`ostream& ostream::operator<<(float n)`

`ostream& ostream::operator<<(double n)`

`ostream& ostream::operator<<(long double n)`

`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。

上記以外の場合、`badbit` を設定します。

リターン値は`*this` です。

`ostream& ostream::write(const char* s, streamsize n)`

`ostream& ostream::write(const signed char* s, streamsize n)`

`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。

上記以外の場合、`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	<code>ostream& endl(ostream& os)</code>	改行を挿入し、出力ストリームをフラッシュします。
	<code>ostream& ends(ostream& os)</code>	ヌルコードを挿入します。
	<code>ostream& flush(ostream& os)</code>	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 signed char* s)	
	ostream& operator<<(ostream& os, const unsigned char* s)	

ostream& operator<<(ostream& os, char s)

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 signed char* s)

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。

上記以外の場合、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	<code>smanip resetiosflags(ios_base::fmtflags mask)</code>	mask 値で指定されたフラグをクリアします。
	<code>smanip setiosflags(ios_base::fmtflags mask)</code>	書式フラグ(fmtfl)を設定します。
	<code>smanip setbase(int base)</code>	出力時に用いる基数を設定します。
	<code>smanip setfill(char c)</code>	詰め文字(fillch)を設定します。
	<code>smanip setprecision(int n)</code>	精度(prec)を設定します。
	<code>smanip setw(int n)</code>	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。

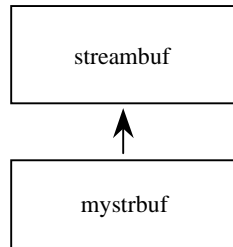
`smanip setprecision(int n)`
 精度(prec)を設定します。
 リターン値は、入出力対象のオブジェクトです。

`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。
	mystrbuf(void* 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* open(const 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_tye* fp)	初期処理です。

例：

```
#include <istream>
#include <ostream>
#include <mystrbuf>
#include <string>
#include <new>
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;
}
```

(3) 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：

- 割当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	<code>new_handler</code>	void 型を返す関数へのポインタ型です。
変数	<code>_ec2p_new_handler</code>	例外処理関数へのポインタです。
関数	<code>void* operator new(size_t size)</code>	size 分の領域を確保します。
	<code>void* operator new[](size_t size)</code>	size 分の配列領域を確保します。
	<code>void* operator new(size_t size, void* ptr)</code>	ptr の指している領域を記憶領域として割り当てます。
	<code>void* operator new[](size_t size, void* ptr)</code>	ptr の指している領域を配列領域として割り当てます。
	<code>void operator delete(void* ptr)</code>	領域を解放します。
	<code>void operator delete[](void* ptr)</code>	配列領域を解放します。
	<code>new_handler set_new_handler(new_handler new_P)</code>	<code>_ec2p_new_handler</code> 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。

(4) 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、和を lhs に格納します。
	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、差を lhs に格納します。
	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、積を lhs に格納します。
	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、商を lhs に格納します。
	float_complex operator/(const float_complex& lhs, const float& rhs)	
	float_complex 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)	

10. C/C++言語仕様

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)	
	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, const 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_complex& 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。
	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)`

`float_complex operator+(const float_complex& lhs, const float& rhs)`

`float_complex operator+(const float& lhs, const float_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、`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)`

`float_complex operator-(const float_complex& lhs, const float& rhs)`

`float_complex operator-(const float& lhs, const float_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、`float_complex(lhs)-=rhs` です。

`float_complex operator*(const float_complex& lhs, const float_complex& rhs)`

`float_complex operator*(const float_complex& lhs, const float& rhs)`

`float_complex operator*(const float& lhs, const float_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、`float_complex(lhs)*=rhs` です。

`float_complex operator/(const float_complex& lhs, const float_complex& rhs)`

`float_complex operator/(const float_complex& lhs, const float& rhs)`

`float_complex operator/(const float& lhs, const float_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、`float_complex(lhs)/=rhs` です。

`bool operator==(const float_complex& lhs, const float_complex& rhs)`

`bool operator==(const float_complex& lhs, const float& rhs)`

`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)`

`bool operator!=(const float_complex& lhs, const float& rhs)`

`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)`

`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)`

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。

<code>float_complex pow(const float_complex& x, const float_complex& y)</code> の場合	: <code>exp(y*logf(x))</code>
上記以外	: <code>exp(y*log(x))</code>

`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	<code>value_type</code>	<code>double</code> 型です。
変数	<code>_re</code>	<code>double</code> 精度の実数部を定義します。
	<code>_im</code>	<code>double</code> 精度の虚数部を定義します。
関数	<code>double_complex(</code> <code>double re = 0.0,</code> <code>double im = 0.0)</code>	コンストラクタです。
	<code>double_complex(const float_complex&)</code>	
	<code>double real() const</code>	実数部を求めます。
	<code>double imag() const</code>	虚数部を求めます。
	<code>double_complex& operator=(double rhs)</code>	<code>rhs</code> を実数部にコピーします。虚数部は 0.0 を設定します。
	<code>double_complex& operator+=(double rhs)</code>	<code>rhs</code> を実数部に加算し、和を <code>*this</code> に格納します。
	<code>double_complex& operator-=(double rhs)</code>	<code>rhs</code> を実数部から減算し、差を <code>*this</code> に格納します。
	<code>double_complex& operator*=(double rhs)</code>	<code>rhs</code> を乗算し、積を <code>*this</code> に格納します。
	<code>double_complex& operator/=(double rhs)</code>	<code>rhs</code> で除算し、商を <code>*this</code> に格納します。
	<code>double_complex& operator=(</code> <code>const double_complex& rhs)</code>	<code>rhs</code> をコピーします。
	<code>double_complex& operator+=(</code> <code>const double_complex& rhs)</code>	<code>rhs</code> を加算し、和を <code>*this</code> に格納します。
	<code>double_complex& operator+=(</code> <code>const double_complex& rhs)</code>	<code>rhs</code> を加算し、和を <code>*this</code> に格納します。
	<code>double_complex& operator*=(</code> <code>const double_complex& rhs)</code>	<code>rhs</code> を乗算し、積を <code>*this</code> に格納します。
	<code>double_complex& operator/=(</code> <code>const double_complex& rhs)</code>	<code>rhs</code> で除算し、商を <code>*this</code> 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	<code>double_complex operator+(const double_complex& lhs)</code>	lhs の単項 + 演算を行います。
	<code>double_complex operator+(const double_complex& lhs, const double_complex& rhs)</code>	lhs と rhs を加算し、和を lhs に格納します。
	<code>double_complex operator+(const double_complex& lhs, const double& rhs)</code>	
	<code>double_complex operator+(const double& lhs, const double_complex& rhs)</code>	
	<code>double_complex operator-(const double_complex& lhs)</code>	lhs の単項 - 演算を行います。
	<code>double_complex operator-(const double_complex& lhs, const double_complex& rhs)</code>	lhs から rhs を減算し、差を lhs に格納します。
	<code>double_complex operator-(const double_complex& lhs, const double& rhs)</code>	
	<code>double_complex operator-(const double& lhs, const double_complex& rhs)</code>	
	<code>double_complex operator*(const double_complex& lhs, const double_complex& rhs)</code>	lhs と rhs を乗算し、積を lhs に格納します。
	<code>double_complex operator*(const double_complex& lhs, const double& rhs)</code>	
	<code>double_complex operator*(const double& lhs, const double_complex& rhs)</code>	
	<code>double_complex operator/(const double_complex& lhs, const double_complex& rhs)</code>	lhs を rhs で除算し、商を lhs に格納します。
	<code>double_complex operator/(const double_complex& lhs, const double& rhs)</code>	
	<code>double_complex operator/(const double& lhs, const double_complex& rhs)</code>	
	<code>bool operator==(const double_complex& lhs, const double_complex& rhs)</code>	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	<code>bool operator==(const double_complex& lhs, const double& rhs)</code>	
	<code>bool operator==(const double& lhs, const double_complex& rhs)</code>	

種別	定義名	説明
関数	<code>bool operator!=(const double_complex& lhs, const double_complex& rhs)</code>	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	<code>bool operator!=(const double_complex& lhs, const double& rhs)</code>	
	<code>bool operator!=(const double& lhs, const double_complex& rhs)</code>	
	<code>istream& operator>>(istream& is, double_complex& x)</code>	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。
	<code>ostream& operator<<(ostream& os, const double_complex& x)</code>	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。
	<code>double real(const double_complex& x)</code>	実数部を求めます。
	<code>double imag(const double_complex& x)</code>	虚数部を求めます。
	<code>double abs(const double_complex& x)</code>	絶対値を求めます。
	<code>double arg(const double_complex& x)</code>	位相角度を求めます。
	<code>double norm(const double_complex& x)</code>	2乗の絶対値を求めます。
	<code>double_complex conj(const double_complex& x)</code>	共役複素数を求めます。
	<code>double_complex polar(const double& rho, const double& theta)</code>	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。
	<code>double_complex cos(const double_complex& x)</code>	複素余弦を求めます。
	<code>double_complex cosh(const double_complex& x)</code>	複素双曲余弦を求めます。
	<code>double_complex exp(const double_complex&)</code>	指数関数を求めます。
	<code>double_complex log(const double_complex& x)</code>	自然対数を求めます。
	<code>double_complex log10(const double_complex& x)</code>	常用対数を求めます。
	<code>double_complex pow(const double_complex& x, int y)</code>	x の y 乗を求めます。
	<code>double_complex pow(const double_complex& x, const double & y)</code>	
	<code>double_complex pow(const double_complex& x, const double_complex& y)</code>	
	<code>double_complex pow(const double & x, const double_complex& y)</code>	
	<code>double_complex sin(const double_complex& x)</code>	複素正弦を求めます。

種別	定義名	説明
関数	<code>double_complex sinh(const double_complex& x)</code>	複素双曲正弦を求めます。
	<code>double_complex sqrt(const double_complex& x)</code>	右半空間における範囲での平方根を求めます。
	<code>double_complex tan(const double_complex& x)</code>	複素正接を求めます。
	<code>double_complex tanh(const double_complex& x)</code>	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)`

`double_complex operator+(const double_complex& lhs, const double& rhs)`

`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)`

`double_complex operator-(const double_complex& lhs, const double& rhs)`

`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)`

`double_complex operator*(const double_complex& lhs, const double& rhs)`

`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)`

`double_complex operator/(const double_complex& lhs, const double& rhs)`

`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)`

`bool operator==(const double_complex& lhs, const double& rhs)`

`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)`

`bool operator!=(const double_complex& lhs, const double& rhs)`

`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)`

`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。

`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。

(5) 文字列操作クラスライブラリ

文字列操作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。
	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。
関数	string(void)	コンストラクタです。
	string::string(const string& str, size_t pos = 0, size_t n = npos)	
	string::string(const char* str, size_t n)	
	string::string(const char* str)	
	string::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長さを n に変更します。
	void resize(size_t n)	格納可能な文字列の長さを 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長さが 0 がチェックします。
	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)	

種別	定義名	説明
関数	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)	
	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)	

10. C/C++言語仕様

種別	定義名	説明
関数	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。
	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。

種別	定義名	説明
関数	<code>size_t find(char c, size_t pos = 0) const</code>	位置 <code>pos</code> 以降で文字 <code>c</code> が最初に現れる位置を検索します。
	<code>size_t rfind(const string& str, size_t pos = npos) const</code>	位置 <code>pos</code> 以前で <code>str</code> 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。
	<code>size_t rfind(const char* str, size_t pos = npos) const</code>	
	<code>size_t rfind(const char* str, size_t pos, size_t n) const</code>	位置 <code>pos</code> 以前で <code>str</code> の <code>n</code> 文字分と同じ文字列が最後に現れる位置を検索します。
	<code>size_t rfind(char c, size_t pos = npos) const</code>	位置 <code>pos</code> 以前で文字 <code>c</code> が最後に現れる位置を検索します。
	<code>size_t find_first_of(const string& str, size_t pos = 0) const</code>	位置 <code>pos</code> 以降で文字列 <code>str</code> 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。
	<code>size_t find_first_of(const char* str, size_t pos = 0) const</code>	
	<code>size_t find_first_of(const char* str, size_t pos, size_t n) const</code>	位置 <code>pos</code> 以降で文字列 <code>str</code> の <code>n</code> 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。
	<code>size_t find_first_of(char c, size_t pos = 0) const</code>	位置 <code>pos</code> 以降で文字 <code>c</code> が最初に現れる位置を検索します。
	<code>size_t find_last_of(const string& str, size_t pos = npos) const</code>	位置 <code>pos</code> 以前で文字列 <code>str</code> 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。
	<code>size_t find_last_of(const char* str, size_t pos = npos) const</code>	
	<code>size_t find_last_of(const char* str, size_t pos, size_t n) const</code>	位置 <code>pos</code> 以前で文字列 <code>str</code> の <code>n</code> 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。
	<code>size_t find_last_of(char c, size_t pos = npos) const</code>	位置 <code>pos</code> 以前で文字 <code>c</code> が最後に現れる位置を検索します。
	<code>size_t find_first_not_of(const string& str, size_t pos = 0) const</code>	位置 <code>pos</code> 以降で <code>str</code> 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。
	<code>size_t find_first_not_of(const char* str, size_t pos = 0) const</code>	
	<code>size_t find_first_not_of(const char* str, size_t pos, size_t n) const</code>	位置 <code>pos</code> 以降で <code>str</code> の先頭から <code>n</code> 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。
	<code>size_t find_first_not_of(char c, size_t pos = 0) const</code>	位置 <code>pos</code> 以降で文字 <code>c</code> と異なった文字が最初に現れる位置を検索します。

10. C/C++言語仕様

種別	定義名	説明
関数	<pre>size_t find_last_not_of(const string& str, size_t pos = npos) const</pre>	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。
	<pre>size_t find_last_not_of(const char* str, size_t pos = npos) const</pre>	
	<pre>size_t find_last_not_of(const char* str, size_t pos, size_t n) const</pre>	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。
	<pre>size_t find_last_not_of(char c, size_t pos = npos) const</pre>	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。
	<pre>string substr(size_t pos = 0, size_t n = npos) const</pre>	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。
	<pre>int compare(const string& str) const</pre>	文字列と str の文字列を比較します。
	<pre>int compare(size_t pos1, size_t n1, const string& str) const</pre>	位置 pos1 から n1 文字分の文字列と str を比較します。
	<pre>int compare(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2) const</pre>	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。
	<pre>int compare(const char* str) const</pre>	str と比較します。
	<pre>int compare(size_t pos1, size_t n1, const char* str, size_t n2 = npos) const</pre>	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()`

`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()`

`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`

`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長さを `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長さを `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 `clear` します。

`bool string::empty() const`

格納している文字列の長さが `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
char& string::operator[](size_t pos)
const char& string::at(size_t pos) const
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)
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)
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)

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)

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

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`

`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`

`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`

`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`

`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`

`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`

`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)	
	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)
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)
```

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)
bool operator==(const char* lhs, const string& rhs)
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 : true

文字列が異なる場合 : false

```
bool operator!=(const string& lhs, const string& rhs)
bool operator!=(const char* lhs, const string& rhs)
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)
bool operator<(const char* lhs, const string& rhs)
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)
bool operator>(const char* lhs, const string& rhs)
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)`

`bool operator<=(const char* lhs, const string& rhs)`

`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)`

`bool operator>=(const char* lhs, const string& rhs)`

`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。

10.3.3 リエントラントライブラリ

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 10.40 にリエントラントライブラリ一覧を示します。表中、`x` 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 x:ノンリエントラント :`errno` を設定
表 10.40 リエントラントライブラリ一覧

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
<code>stddef.h</code>	<code>offsetof</code>		<code>mathf.h</code>	<code>atan2f</code>	
<code>assert.h</code>	<code>assert</code>	x		<code>cosf</code>	
<code>ctype.h</code>	<code>isalnum</code>			<code>sinf</code>	
	<code>isalpha</code>			<code>tanf</code>	
	<code>iscntrl</code>			<code>coshf</code>	
	<code>isdigit</code>			<code>sinhf</code>	
	<code>isgraph</code>			<code>tanhf</code>	
	<code>islower</code>			<code>expf</code>	
	<code>isprint</code>			<code>frexpf</code>	
	<code>ispunct</code>			<code>ldexpf</code>	
	<code>isspace</code>			<code>logf</code>	
	<code>isupper</code>			<code>log10f</code>	
	<code>isxdigit</code>			<code>modff</code>	
	<code>toupper</code>			<code>powf</code>	
	<code>toupper</code>			<code>sqrtf</code>	
<code>math.h</code>	<code>acos</code>			<code>ceilf</code>	
	<code>asin</code>			<code>fabsf</code>	
	<code>atan</code>			<code>floorf</code>	
	<code>atan2</code>			<code>fmodf</code>	
	<code>cos</code>		<code>setjmp.h</code>	<code>setjmp</code>	
	<code>sin</code>			<code>longjmp</code>	
	<code>tan</code>		<code>stdarg.h</code>	<code>va_start</code>	
	<code>cosh</code>			<code>va_arg</code>	
	<code>sinh</code>			<code>va_end</code>	
	<code>tanh</code>		<code>stdio.h</code>	<code>fclose</code>	x
	<code>exp</code>			<code>fflush</code>	x
	<code>frexp</code>			<code>fopen</code>	x
	<code>ldexp</code>			<code>freopen</code>	x
	<code>log</code>			<code>setbuf</code>	x
	<code>log10</code>			<code>setvbuf</code>	x
	<code>modf</code>			<code>fprintf</code>	x
	<code>pow</code>			<code>fscanf</code>	x
	<code>sqrt</code>			<code>printf</code>	x
	<code>ceil</code>			<code>scanf</code>	x
	<code>fabs</code>			<code>sprintf</code>	
	<code>floor</code>			<code>sscanf</code>	
	<code>fmod</code>			<code>vfprintf</code>	x
<code>mathf.h</code>	<code>acosf</code>			<code>vprintf</code>	x
	<code>asinf</code>			<code>vsprintf</code>	
	<code>atanf</code>			<code>fgetc</code>	x

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
stdio.h	fgets	x	stdlib.h	malloc	x
	fputc	x		realloc	x
	fputs	x		bsearch	
	getc	x		qsort	
	getchar	x		abs	
	gets	x		div	
	putc	x		labs	
	putchar	x		ldiv	
	puts	x	string.h	memcpy	
	ungetc	x		strcpy	
	fread	x		strncpy	
	fwrite	x		strcat	
	fseek	x		strncat	
	ftell	x		memcmp	
	rewind	x		strcmp	
	clearerr	x		strncmp	
	feof	x		memchr	
	ferror	x		strchr	
	perror	x		strcspn	
	stdlib.h	atof			strpbrk
		atoi		strchr	
atol			strspn		
strtod			strstr		
strtol			strtok	x	
rand		x	memset		
srand		x	strerror		
calloc		x	strlen		
free		x	memmove		

10.3.4 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10.41に示します。

表 10.41 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	strtoul、abort、atexit、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

【注】*1 ヘッダファイルをサポートしません。

10.3.5 DSP ライブラリ

(1) 概要

SH2-DSP および SH3-DSP(以降、合わせて単に SH-DSP と呼ぶこととします)で使用できるデジタル信号処理(DSP)ライブラリについて説明します。本ライブラリは標準的な DSP 関数を含んでおり、単独または連続的に使用することによって、DSP 演算を行うことができます。

本ライブラリは以下の関数を含んでいます。

- 高速フーリエ変換
- 窓関数
- フィルタ
- 畳み込みと相関
- その他

本ライブラリ関数は高速フーリエ変換とフィルタを除いてリエントラントです。

本ライブラリを使用するときは、表 10.42 に示すファイルをインクルードしてください。また、表 10.43 に示すように、CPU およびコンパイルオプションに対応するライブラリをリンクしてください。

本ライブラリを呼び出した際、関数が正常に終了した場合は EDSP_OK を、異常があった場合は EDSP_BAD_ARG もしくは EDSP_NO_HEAP をリターン値として返します。リターン値の詳細については各関数の説明を参照してください。

表 10.42 DSP ライブラリ用のインクルードファイル

ライブラリの種類	内容	インクルードファイル
1 DSP ライブラリ	DSP 演算を行うライブラリです。	<ensigdsp.h> ----- <filt_ws.h> ^{*1}

【注】*1 フィルタ関数を使用する場合、ユーザプログラム内で 1 回のみインクルードしてください。

表 10.43 DSP ライブラリ一覧

CPU	オプション	ライブラリ名
SH2-DSP	-pic=0	shdsplib.lib
	-pic=1	shdspic.lib
SH3-DSP	-pic=0 -endian=big	sh3dspnb.lib
	-pic=1 -endian=big	sh3dspnb.lib
	-pic=0 -endian=little	sh3dspnl.lib
	-pic=1 -endian=little	sh3dspnl.lib

(2) データフォーマット

本ライブラリはデータを符号付き 16 ビット固定小数点数として扱います。符号付き 16 ビット固定小数点数は図 10.7(a)に示すように、小数点が最上位ビット(MSB)の右側に固定されたデータフォーマットとなっており、 $-1 \sim 1 \cdot 2^{-15}$ の範囲の値を表現できます。

本ライブラリでは、データの受け渡しは short 型のデータフォーマットを使用します。したがって、C/C++ プログラムから本ライブラリを使用する場合、データを符号付き 16 ビット固定小数点数で表現する必要があります。

(例) +0.5 は符号付き 16 ビット固定小数点数で表現すると H'4000 です。したがってライブラリ関数に渡す short 型実引数は H'4000 となります。

本ライブラリ内部の演算では、符号付き 32 ビット固定小数点数と符号付き 40 ビット固定小数点数

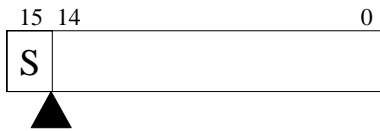
も使用します。符号付き 32 ビット固定小数点数は図 10.7(b)に示すデータフォーマットとなっており、 $-1 \sim 1 \cdot 2^{-31}$ の範囲の値を表現できます。符号付き 40 ビット固定小数点数は図 10.7(c)に示すように 8 ビットのガードビットが付加されたデータフォーマットとなっており、 $-2^8 \sim 2^8 \cdot 2^{-31}$ の範囲の値を表現できます。

符号付き 16 ビット固定小数点数の乗算結果は符号付き 32 ビット固定小数点数で保持します。DSP 命令を用いた固定小数点乗算では、 $H^*8000 \times H^*8000$ の場合だけオーバーフローが発生することに注意してください。また乗算結果の最下位ビット(LSB)は常に 0 になります。乗算結果を次の演算に使用する場合、上位 16 ビットを取り出し、符号付き 16 ビット固定小数点数に変換します。このときアンダフローや精度低下が発生する可能性があります。

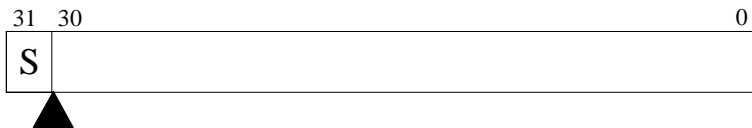
本ライブラリの積和演算では、加算結果を符号付き 40 ビット固定小数点数で保持します。加算のときにオーバーフローが発生しないように注意してください。

演算の際、オーバーフローが発生すると正しい結果が得られません。オーバーフローを防ぐためには、係数や入力データをスケールする必要があります。本ライブラリには、スケールの機能が組み込まれています。スケールの詳細については各関数の説明を参照してください。

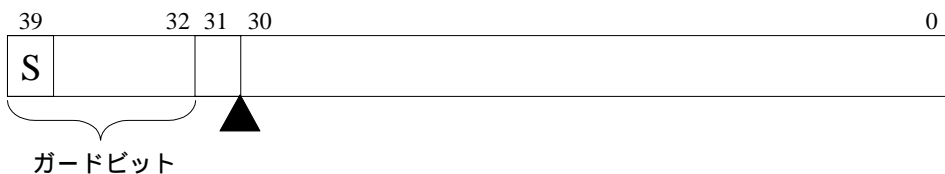
- (a) 符号付き16ビット固定小数点数
($-1 \sim 1 \cdot 2^{-15}$)



- (b) 符号付き32ビット固定小数点数
($-1 \sim 1 \cdot 2^{-31}$)



- (c) 符号付き40ビット固定小数点数
($-2^8 \sim 2^8 \cdot 2^{-31}$)



S : 符号ビット
: 小数点

図 10.7 データフォーマット

(3) 効率

本ライブラリ関数は、SH-DSP上で高速に実行するように最適化しています。

ライブラリを効率よく活用するために、開発するシステムのメモリマップを決める際には、できるだけ以下の2つの推奨事項に従ってください。

- プログラムコードセグメントは、1サイクルでの32ビットリードをサポートしているメモリに配置する。
- データセグメントは、1サイクルでの16(または32)ビットリード・ライトをサポートしているメモリに配置する。

使用するマイコンが、ライブラリコードとデータを配置するのに十分な容量の32ビットメモリを内蔵している場合は、その32ビットメモリに配置するのが最適です。その他のメモリを使用しなければならない場合は、可能な限り上記の推奨事項に従ってください。

(4) 高速フーリエ変換

(a) 関数一覧

関数	説明
FftComplex	not-in-place 複素数 FFT を実行します。
FftReal	not-in-place 実数 FFT を実行します。
IfftComplex	not-in-place 複素数逆 FFT を実行します。
IfftReal	not-in-place 実数逆 FFT を実行します。
FftInComplex	in-place 複素数 FFT を実行します。
FftInReal	in-place 実数 FFT を実行します。
IfftInComplex	in-place 複素数逆 FFT を実行します。
IfftInReal	in-place 実数逆 FFT を実行します。
LogMagnitude	複素数データを対数絶対値に変換します。
InitFft	FFT 回転係数を生成します。
FreeFft	FFT 回転係数の格納に使用したメモリを解放します。

【注】 not-in-place、in-place については「(4)(e) FFT 構造」を参照してください。

これらの関数は、ユーザが定義したスケーリングを使って、順方向高速フーリエ変換と逆方向高速フーリエ変換を実行します。

順方向フーリエ変換は以下の式で定義されます。

$$y_n = 2^{-s} \sum_{m=0}^N e^{-2j\pi m/N} \cdot X_m$$

ここで、s はスケーリングが行われるステージの数、N はデータ数を示しています。

逆方向フーリエ変換は以下の式で定義されます。

$$y_n = 2^{-s} \sum_{m=0}^N e^{2j\pi m/N} \cdot X_m$$

スケーリングについては「(4)(d) スケーリング」を参照してください。

(b) 複素数データ配列フォーマット

FFT および IFFT の複素数データ配列は、実数を X メモリに、虚数を Y メモリに配置します。ただし、実数 FFT の出力データと実数 IFFT の入力データの配置は異なります。実数、虚数を格納する配列をそれぞれ x,y とすると、x[0]には DC 成分の実数成分が入り、y[0]には DC 成分の虚数成分ではなく Fs/2 成分の実数成分が入ります(DC 成分と Fs/2 成分はどちらも実数で、虚数成分は 0 です)。

(c) 実数データ配列フォーマット

FFT および IFFT の実数データ配列フォーマットには、以下の 3 種類があります。

- 単一の配列に格納し、任意のメモリブロックに配置。
- 単一の配列に格納し、X メモリに配置。
- 2つの配列に分けて格納。それぞれの配列のサイズは N/2 で、配列の前半は X メモリに配置し、後半は Y メモリに配置。

FftReal は 1 番目の指定方法のみです。IfftReal、FftInReal および IfftInReal は 2 番目か 3 番目の方法をユーザが選択します。

(d) スケーリング

基数 2 の FFT は各ステージで信号強度が倍になり、ピーク信号振幅も倍になります。そのため、高強度信号を変換する際にオーバーフローが発生することがありますが、各ステージで信号を 1/2 にすることにより(これをスケーリングといいます)オーバーフローを防ぐことができます。しかし、スケーリングしすぎると不要な量子化雑音が発生する可能性があります。

オーバーフローや量子化雑音とスケーリングの最適なバランスは入力信号の特性に大きく依存します。スペクトルが大きなピークを持つ信号はオーバーフローを防ぐために最大スケーリングが必要になりますが、インパルス信号ではスケーリングの必要はほとんどありません。

すべてのステージでスケーリングするのが最も安全な方法です。入力データが強度 2^{30} 未満であれば、この方法でオーバーフローを防ぐことができます。本ライブラリでは、各ステージごとにスケーリングを行うかどうかを指定できます。したがって、スケーリング指定を精密に行うことによって、オーバーフローと量子化雑音の影響を最小限に抑えることができます。

スケーリングの方法を指定するために、各 FFT 関数の引数に `scale` が含まれています。`scale` は最下位ビットから 1 ビットずつが各ステージに対応しています。対応する `scale` のビットが 1 に設定されているすべてのステージで、2 の除算を実行します。

本ライブラリは実行速度を上げるために基数 4 の FFT を使用しています。`scale` は最下位ビットから 2 ビットずつが各ステージに対応しています。どちらか 1 ビットが 1 に設定されていれば、2 の除算を実行します。両方が 1 に設定されていれば 4 の除算を実行します。つまり、2 つの基数 2 の FFT ステージが 1 つの基数 4 の FFT ステージに置き換えられたのと同じこととなります。しかし、基数 2 の FFT よりも基数 4 の FFT の方が量子化雑音の発生する可能性があります。

以下に `scale` の例を示します。

- `scale = H'FFFFFFFF`(または `size-1`)はすべての基数 2 の FFT ステージでスケーリングを行います。すべての入力データの強度が 2^{30} 未満であれば、オーバーフローは発生しません。
- `scale = H'55555555` は 1 つおきの基数 2 の FFT ステージでスケーリングを行います。
- `scale = 0` はスケーリングを行いません。

これらの `scale` の値は、`ensigdsp.h` で `EFFTALLSCALE(H'FFFFFFFF)`、`EFFTMIDSCALE(H'55555555)`、`EFFTNOSCALE(0)`と定義されています。

(e) FFT 構造

本ライブラリの FFT 構造には not-in-place FFT と in-place FFT の 2 種類があります。

not-in-place FFT では、入力データを RAM から取り出し、FFT を実行し、出力結果を RAM のユーザが指定した別の場所に格納します。

一方 in-place FFT では、入力データを RAM から取り出し、FFT を実行し、出力結果を RAM の同じ場所に格納します。この方法を用いると FFT の実行時間は増加しますが、使用メモリスペースが削減できます。

入力データを FFT 関数の他にも使用する場合は、not-in-place FFT を使用してください。また、メモリスペースを節約したい場合は、in-place FFT を使用してください。

not-in-place 複素数 FFT

***int FftComplex (short op_x[], short op_y[], const short ip_x[],
const short ip_y[], long size, long scale)***

説 明 複素数高速フーリエ変換を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 • size < 4
 • size が 2 の累乗ではありません
 • size > max_fft_size

引 数 op_x[] 出力データの実数成分
 op_y[] 出力データの虚数成分
 ip_x[] 入力データの実数成分
 ip_y[] 入力データの虚数成分
 size FFT のサイズ
 scale スケーリング指定

備 考 本関数は *not-in-place* で行いますので、入力配列と出力配列を別々に用意してください。複素数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」を参照してください。
 本関数を呼び出す前に *InitFft* を呼び出して、回転係数と *max_fft_size* を初期化してください。
 スケーリング指定については「(4)(d) スケーリング」を参照してください。
scale は下位 $\log_2(\text{size})$ ビットを使用します。
 本関数はリエントラントではありません。

int FftReal (short op_x[], short op_y[], const short ip[], long size, long scale)

説明 実数高速フーリエ変換を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ size < 8
・ size が 2 の累乗ではありません
・ size > max_fft_size

引数 op_x[] 正の出力データの実数成分
op_y[] 正の出力データの虚数成分
ip[] 実数入力データ
size FFT のサイズ
scale スケーリング指定

備考 op_x と op_y には size/2 の正の出力データが格納されます。負の出力データは正の出力データの共役複素数です。また、0 と $F_s/2$ での出力データの値は実数なので、 $F_s/2$ での実数出力は op_y[0] に格納されます。

本関数は not-in-place で行いますので、入力配列と出力配列を別々に用意してください。複素数と実数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」「(4)(c) 実数データ配列フォーマット」を参照してください。

本関数を呼び出す前に InitFft を呼び出して、回転係数と max_fft_size を初期化してください。

スケーリング指定については「(4)(d) スケーリング」を参照してください。

scale は下位 $\log_2(\text{size})$ ビットを使用します。

本関数はリエントラントではありません。

not-in-place 複素数逆FFT

***int IfftComplex (short op_x[], short op_y[], const short ip_x[],
const short ip_y[], long size, long scale)***

説明 複素数逆高速フーリエ変換を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・size < 4
・size が 2 の累乗ではありません
・size > max_fft_size

引数 op_x[] 出力データの実数成分
op_y[] 出力データの虚数成分
ip_x[] 入力データの実数成分
ip_y[] 入力データの虚数成分
size 逆FFTのサイズ
scale スケーリング指定

備考 本関数は *not-in-place* で行いますので、入力配列と出力配列を別々に用意してください。複素数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」を参照してください。
本関数を呼び出す前に *InitFft* を呼び出して、回転係数と *max_fft_size* を初期化してください。
スケーリング指定については「(4)(d) スケーリング」を参照してください。
scale は下位 $\log_2(\text{size})$ ビットを使用します。
本関数はリエントラントではありません。

not-in-place 実数逆FFT

***int IfftReal (short op_x[], short scratch_y[], const short ip_x[],
const short ip_y[], long size, long scale, int op_all_x)***

説明 実数逆高速フーリエ変換を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です

- size < 8
- size が 2 の累乗ではありません
- size > max_fft_size
- op_all_x ≠ 0 または 1

引数 op_x[] 実数出力データ
scratch_y[] スクラッチメモリまたは実数出力データ
ip_x[] 正の入力データの実数成分
ip_y[] 正の入力データの虚数成分
size 逆FFTのサイズ
scale スケーリング指定
op_all_x 出力データの配置指定

備考 ip_x と ip_y には size/2 の正の入力データを格納してください。負の入力データは正の入力データの共役複素数です。また、0 と $F_s/2$ での入力データの値は実数なので、 $F_s/2$ での実数入力はいp_y[0]に格納してください。
出力データのフォーマットは op_all_x で指定します。op_all_x=1 の場合、全出力データは op_x に格納されます。op_all_x=0 の場合、最初の size/2 の出力データは op_x に格納され、残りの size/2 の出力データは scratch_y に格納されます。
本関数は not-in-place で行いますので、入力配列と出力配列を別々に用意してください。複素数と実数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」「(4)(c) 実数データ配列フォーマット」を参照してください。
ip_x、ip_y はそれぞれ size/2 のデータを格納してください。op_x は op_all_x の値によって、size または size/2 のデータが格納されます。
本関数を呼び出す前に InitFft を呼び出して、回転係数と max_fft_size を初期化してください。
スケーリング指定については「(4)(d) スケーリング」を参照してください。
scale は下位 $\log_2(\text{size})$ ビットを使用します。
本関数はリエントラントではありません。

in-place 複素数FFT***int FftInComplex (short data_x[], short data_y[], long size, long scale)***

説明 in-place 複素数高速フーリエ変換を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ size < 4
 ・ size が 2 の累乗ではありません
 ・ size > max_fft_size

引数 data_x[] 入出力データの実数成分
 data_y[] 入出力データの虚数成分
 size FFTのサイズ
 scale スケーリング指定

備考 複素数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」を参照してください。
 本関数を呼び出す前に InitFft を呼び出して、回転係数と max_fft_size を初期化してください。
 スケーリング指定については「(4)(d) スケーリング」を参照してください。
 scale は下位 $\log_2(\text{size})$ ビットを使用します。
 本関数はリエントラントではありません。

int FftInReal (short data_x[], short data_y[], long size, long scale, int ip_all_x)

説明 in-place 実数高速フーリエ変換を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ size < 8
 ・ size が 2 の累乗ではありません
 ・ size > max_fft_size
 ・ ip_all_x ≠ 0 または 1

引数 data_x[] 入力時は実数データ、出力時は正の出力データの実数成分
 data_y[] 入力時は実数データまたは未使用、出力時は正の出力データの虚数成分
 size FFT のサイズ
 scale スケーリング指定
 ip_all_x 入力データの配置指定

備考 入力データのフォーマットは、ip_all_x で指定します。ip_all_x=1 の場合、全入力データは data_x から取り出します。ip_all_x=0 の場合、前半の size/2 の入力データは data_x から、後半の size/2 の入力データは data_y から取り出します。
 本関数実行後、data_x と data_y には size/2 の正の出力データが格納されます。負の出力データは正の出力データの共役複素数です。また、0 と $F_s/2$ での出力データの値は実数なので、 $F_s/2$ での実数出力は data_y[0] に格納されます。
 複素数と実数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」「(4)(c) 実数データ配列フォーマット」を参照してください。
 data_y は size/2 のデータを格納します。data_x は ip_all_x の値によって size または size/2 のデータを格納します。
 本関数を呼び出す前に InitFft を呼び出して、回転係数と max_fft_size を初期化してください。
 スケーリング指定については「(4)(d) スケーリング」を参照してください。
 scale は下位 $\log_2(\text{size})$ ビットを使用します。
 本関数はリエントラントではありません。

in-place 複素数逆FFT***int IfftInComplex (short data_x[], short data_y[], long size, long scale)***

説明	in-place 複素数逆高速フーリエ変換を実行します。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK EDSP_BAD_ARG	成功 以下のいずれかの場合です ・size < 4 ・size が 2 の累乗ではありません ・size > max_fft_size
引数	data_x[] data_y[] size scale	入出力データの実数成分 入出力データの虚数成分 逆FFTのサイズ スケーリング指定
備考	<p>複素数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」を参照してください。</p> <p>本関数を呼び出す前に InitFft を呼び出して、回転係数と max_fft_size を初期化してください。</p> <p>スケーリング指定については「(4)(d) スケーリング」を参照してください。</p> <p>scale は下位 $\log_2(\text{size})$ ビットを使用します。</p> <p>本関数はリエントラントではありません。</p>	

int IfftInReal (short data_x[], short data_y[], long size, long scale, int op_all_x)

説明 in-place 実数逆高速フーリエ変換を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ size < 8
・ size が 2 の累乗ではありません
・ size > max_fft_size
・ op_all_x ≠ 0 または 1

引数 data_x[] 入力時は正の入力データの実数成分、出力時は実数データ
data_y[] 入力時は正の入力データの虚数成分、出力時は実数データまたは未使用
size 逆FFTのサイズ
scale スケーリング指定
op_all_x 出力データの配置指定

備考 data_x と data_y には size/2 の正の入力データを格納してください。負の入力データは正の入データの共役複素数です。また、0 と $F_s/2$ での入力データの値は実数なので、 $F_s/2$ での実数入力 は data_y[0] に格納してください。
出力データのフォーマットは op_all_x で指定します。op_all_x=1 の場合、全出力データは data_x に格納されます。op_all_x=0 の場合、前半の size/2 の出力データは data_x に格納され、後半の size/2 の出力データは data_y に格納されます。
複素数と実数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」「(4)(c) 実数データ配列フォーマット」を参照してください。
data_y は size/2 のデータを格納します。data_x は、op_all_x の値によって size または size/2 のデータが格納されます。
本関数を呼び出す前に InitFft を呼び出して、回転係数と max_fft_size を初期化してください。
スケーリング指定については「(4)(d) スケーリング」を参照してください。
scale は下位 $\log_2(\text{size})$ ビットを使用します。
本関数はリエントラントではありません。

対数絶対値

***int LogMagnitude (short output[], const short ip_x[], const short ip_y[],
long no_elements, float fscale)***

説明 複素数入力データの対数絶対値をデシベル単位で計算し、スケーリング結果を出力配列に書き込みます。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ no_elements < 1
・ no_elements > 32767
・ |fscale| ≥ 2¹⁵ / (10log₁₀2³¹)

引数 output[] 実数出力 z
ip_x[] 入力の実数成分 x
ip_y[] 入力の虚数成分 y
no_elements 出力データ数 N
fscale 出力スケーリング係数

備考 $z(n) = 10fscale \cdot \log_{10}(x(n)^2 + y(n)^2)$ $0 \leq n < N$
複素数データ配列の配置については「(4)(b) 複素数データ配列フォーマット」を参照してください。

int InitFft (long max_size)

説明	FFT 関数で使用する回転係数 (1/4 サイズ) を生成します。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK EDSP_NO_HEAP EDSP_BAD_ARG	成功 malloc で確保できるメモリスペースが不十分 以下のいずれかの場合です ・max_size < 2 ・max_size が 2 の累乗ではありません ・max_size > 32768
引数	max_size	必要になる FFT の最大サイズ
備考	<p>回転係数は malloc によって確保されるメモリに格納されます。 回転係数が生成されると max_fft_size グローバル変数が更新されます。max_fft_size は FFT の最大許容サイズを示します。 本関数は最初の FFT 関数を呼び出す前に必ず一度呼び出してください。 max_size は 8 以上としてください。 回転係数は max_size で指定した変換サイズで生成されます。max_size より小さいサイズの FFT 関数を実行したときも同じ回転係数を使用します。 回転係数のアドレスは内部変数内に格納されています。ここはユーザプログラムでアクセスしないでください。 本関数はリエントラントではありません。</p>	

void FreeFft (void)

説明	回転係数の格納に使用したメモリを解放します。
ヘッダ	<ensigdsp.h>
備考	<p>max_fft_size グローバル変数を 0 にします。FreeFft を実行した後再び FFT 関数を実行するときには、その前に必ず InitFft を実行してください。 本関数はリエントラントではありません。</p>

(5) 窓関数

(a) 関数一覧

関数名	説明
GenBlackman	ブラックマン窓を生成します。
GenHamming	ハミング窓を生成します。
GenHanning	ハンニング窓を生成します。
GenTriangle	三角窓を生成します。

ブラックマン窓

int GenBlackman (short output[], long win_size)

説明 ブラックマン窓を生成し、output に出力します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG win_size ≤ 1

引数 output[] 出力データ W(n)
win_size 窓サイズ N

備考 実際のデータにこの窓をかけるときは VectorMult を使用します。
使用する関数を以下に示します。

$$W(n) = (2^{15} - 1) \left[0.42 - 0.5 \cos\left(\frac{2\pi n}{N}\right) + 0.08 \cos\left(\frac{4\pi n}{N}\right) \right] \quad 0 \leq n < N$$

int GenHamming (short output[], long win_size)

説明 ハミング窓を生成し、output に出力します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG win_size ≤ 1

引数 output[] 出力データ W(n)
win_size 窓サイズ N

備考 実際のデータにこの窓をかけるときは VectorMult を使用します。
使用する関数を以下に示します。

$$W(n) = (2^{15} - 1) \left[0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right) \right] \quad 0 \leq n < N$$

int GenHanning (short output[], long win_size)

説明 ハニング窓を生成し、output に出力します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG win_size ≤ 1

引数 output[] 出力データ W(n)
win_size 窓サイズ N

備考 実際のデータにこの窓をかけるときは VectorMult を使用します。
使用する関数を以下に示します。

$$W(n) = \left(\frac{2^{15} - 1}{2} \right) \left[1 - \cos\left(\frac{2\pi n}{N}\right) \right] \quad 0 \leq n < N$$

三角窓

int GenTriangle (short output[], long win_size)

説明 三角窓を生成し、output に出力します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG win_size ≤ 1

引数 output[] 出力データ W(n)
win_size 窓サイズ N

備考 実際のデータにこの窓をかけるときは VectorMult を使用します。
使用する関数を以下に示します。

$$W(n) = (2^{15} - 1) \left[1 - \left| \frac{2n - N + 1}{N + 1} \right| \right] \quad 0 \leq n < N$$

(6) フィルタ

(a) 関数一覧

関数名	説明
Fir	有限インパルス応答フィルタ処理を実行します。
Fir1	単一データ用有限インパルス応答フィルタ処理を実行します。
lir	無限インパルス応答フィルタ処理を実行します。
lir1	単一データ用無限インパルス応答フィルタ処理を実行します。
Dlir	倍精度無限インパルス応答フィルタ処理を実行します。
Dlir1	単一データ用倍精度無限インパルス応答フィルタ処理を実行します。
Lms	適応 FIR フィルタ処理を実行します。
Lms1	単一データ用適応 FIR フィルタ処理を実行します。
InitFir	FIR フィルタ用に作業領域を割り付けます。
Initlir	IIR フィルタ用に作業領域を割り付けます。
InitDlir	DIIR フィルタ用に作業領域を割り付けます。
InitLms	LMS フィルタ用に作業領域を割り付けます。
FreeFir	InitFir で割り付けられた作業領域を解放します。
Freelir	Initlir で割り付けられた作業領域を解放します。
FreeDlir	InitDlir で割り付けられた作業領域を解放します。
FreeLms	InitLms で割り付けられた作業領域を解放します。

【注】 本関数のいずれかを使用する場合、ユーザプログラム内で1回のみ `filt_ws.h` をインクルードしてください。

(b) 係数のスケールリング

フィルタ処理を行うと飽和または量子化雑音が発生する可能性があります。これらはフィルタ係数のスケールリングを行うことによって最小限に抑えることができます。しかし、飽和と量子化雑音の影響をよく考えてスケールリングを行わなければなりません。係数が大きすぎると飽和が、小さすぎると量子化雑音が発生する可能性があります。

FIR(有限インパルス応答)フィルタの場合、以下の式が成り立つようにフィルタ係数を設定すれば飽和は起こりません。

$$\text{coeff}[i] \neq H \cdot 8000 \quad (\text{すべての } i \text{ について})$$

$$|\text{coeff}| < 2^{24}$$

$$\text{res_shift} = 24$$

`coeff` はフィルタ係数、`res_shift` は出力で行われる右シフトのビット数です。

しかし、多くの入力信号の場合、もっと小さい `res_shift` の値(またはもっと大きな `coeff` の値)を使用しても飽和する可能性は少なく、量子化雑音も大幅に削減できます。また入力値に `H*8000` が含まれている可能性があれば、すべての `coeff` の値は `H*8001 ~ H*7FFF` の範囲になるように設定してください。

IIR(無限インパルス応答)フィルタは再帰的な構造になっています。そのため上述したようなスケールリング方法は適していません。

LMS(最小2乗平均)適応フィルタは FIR フィルタと同様です。しかし、係数を適応するとき飽和を引き起こす場合があります。その場合は、係数に `H*8000` を含まないように設定してください。

(c) 作業領域

デジタルフィルタでは、ある処理から次の処理へ保持しておかなければならない情報があります。これらの情報は、最小オーバーヘッドでアクセスすることができるメモリに格納します。本ライブラリでは、Y-RAM 領域を作業領域として使用します。作業領域はフィルタ処理を実行する前に `Init` 関数を呼び出して初期化してください。

作業領域メモリはライブラリ関数によってアクセスされます。なお、ユーザプログラムから作業領域を直接アクセスしないでください。

(d) メモリの使用

SH-DSP を効率よく使うために、フィルタ係数は X メモリに配置してください。入出力データは任意のメモリセグメントに配置することができます。

フィルタ係数は `#pragma section` 命令を用いて X メモリに配置してください。

各フィルタは `Init` 関数を用いてグローバルバッファから作業領域を割り付けます。グローバルバッファは Y メモリに配置します。

int Fir (short output[], const short input[], long no_samples, const short coeff[], long no_co coeffs, int res_shift, short *workspace)

説明 有限インパルス応答(FIR)フィルタ処理を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_samples < 1
 ・ no_co coeffs ≤ 6
 ・ res_shift < 0
 ・ res_shift > 25

引数 output[] 出力データ y
 input[] 入力データ x
 no_samples 入力データの数 N
 coeff[] フィルタ係数 h
 no_co coeffs 係数の数(フィルタの長さ)K
 res_shift 各出力に適用される右シフト
 workspace 作業領域へのポインタ

備考 最新の入力データは作業領域に保持されます。input をフィルタ処理した結果は output に書き込まれます。

$$y(n) = \left[\sum_{k=0}^{K-1} h(k) x(n - k) \right] \cdot 2^{-res_shift}$$

積和演算の結果は 39 ビットで保持されます。出力 y(n) は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

係数のスケーリングについては「(6)(b) 係数のスケーリング」を参照してください。本関数を呼び出す前に InitFir を呼び出し、フィルタの作業領域を初期化してください。output に input と同じ配列を指定した場合、input は上書きされます。本関数はリエントラントではありません。

単一データ用 FIR

```
int Fir1 (short *output, short input, const short coeff[ ], long no_coefifs,
int res_shift, short *workspace)
```

説明 単一データ用に有限インパルス応答 (FIR) フィルタ処理を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ no_coefifs ≤ 6
・ res_shift < 0
・ res_shift > 25

引数 output 出力データ y(n)へのポインタ
input 入力データ x(n)
coeff[] フィルタ係数 h
no_coefifs 係数の数(フィルタの長さ)K
res_shift 各出力に適用される右シフト
workspace 作業領域へのポインタ

備考 最新の入力データは作業領域に保持されます。input をフィルタ処理した結果は*output に書き込まれます。

$$y(n) = \left[\sum_{k=0}^{K-1} h(k) x(n - k) \right] \cdot 2^{-res_shift}$$

積和演算の結果は 39 ビットで保持されます。出力 y(n)は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

係数のスケーリングについては「(6)(b) 係数のスケーリング」を参照してください。

関数を呼び出す前に InitFir を呼び出し、フィルタの作業領域を初期化してください。

本関数はリエントラントではありません。

int Iir (short output[], const short input[], long no_samples, const short coeff[], long no_sections, short *workspace)

説明 無限インパルス応答(IIR)フィルタ処理を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_samples < 1
 ・ no_sections < 1
 ・ a_{0k} < 0
 ・ a_{0k} > 16

引数 output[] 出力データ Y_{K-1}
 input[] 入力データ x₀
 no_samples 入力データの数 N
 coeff[] フィルタ係数
 no_sections 2次フィルタセクションの数 K
 workspace 作業領域へのポインタ

備考 フィルタは、バイカッドという2次フィルタをK個縦列に接続した構成になっています。各バイカッドの出力で付加的なスケールリングが行われます。フィルタ係数は符号付き16ビット固定小数点数で指定します。

各バイカッドの出力は以下の式で与えられます。

$$D_k(n) = [a_{1k}d_k(n-1) + a_{2k}d_k(n-2) + 2^{15}x(n)] \cdot 2^{-15}$$

$$Y_k(n) = [b_{0k}d_k(n) + b_{1k}d_k(n-1) + b_{2k}d_k(n-2)] \cdot 2^{-a_{0k}}$$

k番目のセクションの入力 x_k(n)は、前のセクションの出力 Y_{k-1}(n)です。最初のセクション(k=0)の入力は input から読み込まれます。最後のセクション(k=K-1)の出力は output に書き込まれます。

coeff は係数を以下の順序に設定してください。

$$a_{00}, a_{10}, a_{20}, b_{00}, b_{10}, b_{20}, a_{01}, a_{11}, a_{21}, b_{01}, \dots, b_{2K-1}$$

a_{0k}項はk番目のバイカッドの出力で行われる右シフトのビット数です。

各バイカッドでは飽和演算を32ビットで行います。各バイカッドの出力は15ビットまたはa_{0k}ビット右シフトした結果の下位16ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

本関数を呼び出す前に InitIir を呼び出し、フィルタの作業領域を初期化してください。

output に input と同じ配列を指定した場合、input は上書きされます。

本関数はリエントラントではありません。

単一データ用 IIR

int Iir1 (short *output, short input, const short coeff[], long no_sections, short *workspace)

説明 単一データ用に無限インパルス応答 (IIR) フィルタ処理を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_sections < 1
 ・ a_{ok} < 0
 ・ a_{ok} > 16

引数 output 出力データ Y_{k-1}(n)へのポインタ
 input 入力データ x₀(n)
 coeff[] フィルタ係数
 no_sections 2次フィルタセクションの数 K
 workspace 作業領域へのポインタ

備考 フィルタは、バイカッドという2次フィルタをK個縦列に接続した構成になっています。各バイカッドの出力で付加的なスケールリングが行われます。フィルタ係数は符号付き16ビット固定小数点数で指定します。

各バイカッドの出力は以下の式で与えられます。

$$D_k(n) = [a_{1k}d_k(n-1) + a_{2k}d_k(n-2) + 2^{15}x(n)] \cdot 2^{-15}$$

$$Y_k(n) = [b_{0k}d_k(n) + b_{1k}d_k(n-1) + b_{2k}d_k(n-2)] \cdot 2^{-a_{0k}}$$

k番目のセクションの入力 x_k(n)は、前のセクションの出力 Y_{k-1}(n)です。最初のセクション (k=0)の入力は input から読み込まれます。最後のセクション (k=K-1)の出力は output に書き込まれます。

coeff は係数を以下の順序に設定してください。

a₀₀, a₁₀, a₂₀, b₀₀, b₁₀, b₂₀, a₀₁, a₁₁, a₂₁, b₀₁, ..., b_{2K-1}

a_{0k} 項は k 番目のバイカッドの出力で行われる右シフトのビット数です。

各バイカッドでは飽和演算を32ビットで行います。各バイカッドの出力は15ビットまたは a_{0k}ビット右シフトした結果の下位16ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

本関数を呼び出す前に InitIir を呼び出し、フィルタの作業領域を初期化してください。

本関数はリエントラントではありません。

int Ddir (short output[], const short input[], long no_samples, const long coeff[], long no_sections, long *workspace)

説明 倍精度無限インパルス応答フィルタ処理を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_samples < 1
 ・ no_sections < 1
 ・ a_{0k} < 3
 ・ k < K-1 で a_{0k} > 32
 ・ k = K-1 で a_{0k} > 48

引数 output[] 出力データ Y_{K-1}
 input[] 入力データ x
 no_samples 入力データの数 N
 coeff[] フィルタ係数
 no_sections 2次フィルタセクションの数 K
 workspace 作業領域へのポインタ

備考 フィルタは、バイカッドという2次フィルタをK個縦列に接続した構成になっています。各バイカッドの出力で付加的なスケールが行われます。フィルタ係数は符号付き32ビット固定小数点数で指定します。

各バイカッドの出力は、以下の方程式で与えられます。

$$D_k(n) = [a_{1k}d_k(n-1) + a_{2k}d_k(n-2) + 2^{29}x(n)] \cdot 2^{-31}$$

$$Y_k(n) = [b_{0k}d_k(n) + b_{1k}d_k(n-1) + b_{2k}d_k(n-2)] \cdot 2^{-a_{0k}} \cdot 2^2$$

k番目のセクションの入力 x_k(n)は、前のセクションの出力 Y_{k-1}(n)です。最初のセクション(k=0)の入力は、inputを16ビット左シフトした値が読み込まれます。最後のセクション(k=K-1)の出力はoutputに書き込まれます。

coeffは係数を以下の順序に設定してください。

$$a_{00}, a_{10}, a_{20}, b_{00}, b_{10}, b_{20}, a_{01}, a_{11}, a_{21}, b_{01}, \dots, b_{2K-1}$$

a_{0k}項はk番目のバイカッドの出力で行われる右シフトのビット数です。

DIirは、フィルタ係数を16ビット値ではなく、32ビット値で指定するという点でIirと異なります。積和演算の結果は64ビットで保持されます。中間ステージの出力は、a_{0k}ビット右シフトした結果の下位32ビットが取り出されます。オーバフローしたときは正または負の最大値となります。最終ステージでは、a_{0K-1}ビット右シフトした結果の下位16ビットが取り出されます。なお、オーバフローしたときは正または負の最大値となります。

本関数を呼び出す前にInitDIirを呼び出し、フィルタの作業領域を初期化してください。

遅延ノードd_k(n)は、30ビットの値に丸められ、オーバフローしたときは正または負の最大値となります。

DIirは符号付き32ビット固定小数点数で係数を指定して使用してください。このとき、a_{0k}はk < K-1のときは31、k=K-1のときは47に設定してください。

DIirよりIirの方が実行速度は速いので、倍精度計算の必要がなければIirを使用してください。

outputにinputと同じ配列を指定した場合、inputは上書きされます。

本関数はリエントラントではありません。

単一データ用倍精度 IIR

```
int Ddir1 (short *output, const short input, const long coeff[],
           long no_sections, long *workspace)
```

説明 単一データ用に倍精度無限インパルス応答フィルタ処理を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_sections < 1
 ・ a_{0k} < 3
 ・ k < K-1 で a_{0k} > 32
 ・ k = K-1 で a_{0k} > 48

引数 output 出力データ y_{k-1}(n)へのポインタ
 input 入力データ x₀(n)
 coeff[] フィルタ係数
 no_sections 2次フィルタセクションの数 K
 workspace 作業領域へのポインタ

備考 フィルタは、バイカッドという2次フィルタをK個縦列に接続した構成になっています。各バイカッドの出力で付加的なスケールが行われます。フィルタ係数は符号付き32ビット固定小数点数で指定します。

各バイカッドの出力は、以下の方程式で与えられます。

$$D_k(n) = [a_{1k}d_k(n-1) + a_{2k}d_k(n-2) + 2^{29}x(n)] \cdot 2^{-31}$$

$$Y_k(n) = [b_{0k}d_k(n) + b_{1k}d_k(n-1) + b_{2k}d_k(n-2)] \cdot 2^{-a_{0k}} \cdot 2^2$$

k番目のセクションの入力 x_k(n)は、前のセクションの出力 y_{k-1}(n)です。最初のセクション(k=0)への入力、inputを16ビット左シフトした値が読み込まれます。最後のセクション(k=K-1)からの出力はoutputに書き込まれます。

coeffは係数を以下の順序に設定してください。

$$a_{00}, a_{10}, a_{20}, b_{00}, b_{10}, b_{20}, a_{01}, a_{11}, a_{21}, b_{01}, \dots, b_{2K-1}$$

a_{0k}項はk番目のバイカッドの出力で行われる右シフトのビット数です。

Ddir1は、フィルタ係数を16ビット値ではなく、32ビット値で指定するという点でIir1と異なります。積と演算の結果は64ビットで保持されます。中間ステージの出力は、a_{0k}ビット右シフトした結果の下位32ビットが取り出されます。オーバーフローしたときは正または負の最大値となります。最終ステージでは、a_{0k-1}ビット右シフトした結果の下位16ビットが取り出されます。なお、オーバーフローしたときは正または負の最大値となります。

本関数を呼び出す前にInitDdirを呼び出し、フィルタの作業領域を初期化してください。

遅延ノードd_k(n)は、30ビットの値に丸められ、オーバーフローしたときは正または負の最大値となります。

Ddir1は符号付き32ビット固定小数点数で係数を指定して使用してください。このとき、a_{0k}はk < K-1のときは31、k=K-1のときは47に設定してください。

Ddir1よりIir1の方が実行速度は速いので、倍精度計算の必要がなければIir1を使用してください。

本関数はリエントラントではありません。

***int Lms (short output[], const short input[], const short ref_output[],
long no_samples, short coeff[], long no_co coeffs, int res_shift,
short conv_fact, short *workspace)***

説明 最小 2 乗平均アルゴリズム (LMS) を使って、実数適応 FIR フィルタ処理を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ no_samples < 1
・ no_co coeffs ≤ 6
・ res_shift < 0
・ res_shift > 25

引数 output[] 出力データ y
input[] 入力データ x
ref_output[] 所望の出力値 d
no_samples 入力データの数 N
coeff[] 適応フィルタ係数 h
no_co coeffs 係数の数 K
res_shift 各出力に適用される右シフト
conv_fact 収束係数 2μ
workspace 作業領域へのポインタ

備考 FIR フィルタは以下の式で定義されます。

$$y(n) = \left[\sum_{k=0}^{K-1} h_n(k) x(n-k) \right] \cdot 2^{-res_shift}$$

積和演算の結果は 39 ビットで保持されます。出力 y(n) は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバフローしたときは正または負の最大値となります。

フィルタ係数の更新は Widrow-Hoff アルゴリズムを使用します。

$$h_{n+1}(k) = h_n(k) + 2\mu e(n)x(n-k)$$

ここで e(n) は所望する信号と実際の出力の誤差です。

$$e(n) = d(n) - y(n)$$

2μe(n)x(n-k) の計算では、16 ビット×16 ビットの乗算を 2 回行います。どちらの乗算結果とも上位 16 ビットが保持され、オーバフローしたときは正または負の最大値となります。更新した係数の値が H'8000 になると、積和演算でオーバフローが発生する可能性があります。係数の値は H'8001 ~ H'7FFF の範囲内になるように設定してください。

係数のスケールングについては「(6)(b) 係数のスケールング」を参照してください。

係数は LMS フィルタによって適応させるので、最も安全なスケールングは係数を 256 個未満にし、res_shift を 24 に設定する方法です。

conv_fact は通常正に設定してください。また H'8000 には設定しないでください。

本関数を呼び出す前に InitLms を呼び出し、フィルタを初期化してください。

output に input または ref_output と同じ配列を指定した場合、input または ref_output は上書きされず。

本関数はリエントラントではありません。

単一データ用適応 FIR

int Lms1 (short *output, short input, short ref_output, short coeff[], long no_coeffs, int res_shift, short conv_fact, short *workspace)

説明 最小 2 乗平均アルゴリズム (LMS) を使って、単一データ用に実数適応 FIR フィルタ処理を実行します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_coeffs ≤ 6
 ・ res_shift < 0
 ・ res_shift > 25

引数 output 出力データ $y(n)$ へのポインタ
 input 入力データ $x(n)$
 ref_output 所望の出力値 $d(n)$
 coeff[] 適応フィルタ係数 h
 no_coeffs 係数の数 K
 res_shift 各出力に適用される右シフト
 conv_fact 収束係数 2μ
 workspace 作業領域へのポインタ

備考 FIR フィルタは以下の式で定義されます。

$$y(n) = \left[\sum_{k=0}^{K-1} h_n(k)x(n-k) \right] \cdot 2^{-\text{res_shift}}$$

積和演算の結果は 39 ビットで保持されます。出力 $y(n)$ は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

フィルタ係数の更新は Widrow-Hoff アルゴリズムを使用します。

$$h_{n+1}(k) = h_n(k) + 2\mu e(n)x(n-k)$$

ここで $e(n)$ は所望する信号と実際の出力の誤差です。

$$e(n) = d(n) - y(n)$$

$2\mu e(n)x(n-k)$ の計算では、16 ビット × 16 ビットの乗算を 2 回行います。どちらの乗算とも、上位 16 ビットが保持され、オーバーフローしたときは正または負の最大値となります。更新した係数の値が $H'8000$ になると、積和演算でオーバーフローが発生する可能性があります。係数の値は $H'8001 \sim H'7FFF$ の範囲内になるように設定してください。

係数のスケーリングについては「(6)(b) 係数のスケーリング」を参照してください。

係数は LMS フィルタによって適応させるので、最も安全なスケーリングは係数を 256 個未満にし、 res_shift を 24 に設定する方法です。

conv_fact は通常正に設定してください。また $H'8000$ には設定しないでください。

本関数を呼び出す前に InitLms を呼び出し、フィルタを初期化してください。

本関数はリエントラントではありません。

FIR 作業領域割り付け

int InitFir (short **workspace, long no_coeffs)

説明 Fir と Fir1 で使用する作業領域を割り付けます。

ヘッダ <ensigdsp.h>

リターン値	EDSP_OK	成功
	EDSP_NO_HEAP	workspace の使用できるメモリスペースが不十分
	EDSP_BAD_ARG	no_coeffs ≤ 2

引数	workspace	作業領域へのポインタへのポインタ
	no_coeffs	係数の数 K

備考 すでに入力されているデータは 0 に初期化されます。
Fir、Fir1、Lms および Lms1 だけが InitFir で割り付けられた作業領域を操作することができます。ユーザプログラムから作業領域を直接アクセスしないでください。
本関数はリエントラントではありません。

IIR 作業領域割り付け

int InitIir (short **workspace, long no_sections)

説明 Iir と Iir1 で使用する作業領域を割り付けます。

ヘッダ <ensigdsp.h>

リターン値	EDSP_OK	成功
	EDSP_NO_HEAP	workspace の使用できるメモリスペースが不十分
	EDSP_BAD_ARG	no_sections < 1

引数	workspace	作業領域へのポインタへのポインタ
	no_sections	2次フィルタセクションの数 K

備考 すでに入力されているデータは 0 に初期化されます。
Iir と Iir1 だけが InitIir で割り付けられた作業領域を操作することができます。ユーザプログラムから作業領域を直接アクセスしないでください。
本関数はリエントラントではありません。

倍精度 IIR 作業領域割り付け

int InitDIir (long **workspace, long no_sections)

説明	DIir と DIir1 で使用する作業領域を割り付けます。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK	成功
	EDSP_NO_HEAP	workspace の使用できるメモリスペースが不十分
	EDSP_BAD_ARG	no_sections < 1
引数	workspace	作業領域へのポインタへのポインタ
	no_sections	2 次フィルタセクションの数 K
備考	すでに入力されているデータは 0 に初期化されます。 DIir と DIir1 だけが InitDIir で割り付けられた作業領域を操作することができます。 本関数はリエントラントではありません。	

適応 FIR 作業領域割り付け

int InitLms (short **workspace, long no_coefs)

説明	Lms と Lms1 で使用する作業領域を割り付けます。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK	成功
	EDSP_NO_HEAP	workspace の使用できるメモリスペースが不十分
	EDSP_BAD_ARG	no_coefs ≤ 2
引数	workspace	作業領域へのポインタへのポインタ
	no_coefs	係数の数 K
備考	すでに入力されているデータは 0 に初期化されます。 Fir、Fir1、Lms および Lms1 だけが InitLms で割り付けられた作業領域を操作することができます。 ユーザプログラムから作業領域を直接アクセスしないでください。 本関数はリエントラントではありません。	

int FreeFir (short **workspace, long no_coefs)

説明	InitFir で割り付けられた作業領域を解放します。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK	成功
	EDSP_BAD_ARG	no_coefs ≤ 2
引数	workspace	作業領域へのポインタへのポインタ
	no_coefs	係数の数 K
備考	本関数はリエントラントではありません。	

int FreeIir (short **workspace, long no_sections)

説明	InitIir で割り付けられた作業領域を解放します。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK	成功
	EDSP_BAD_ARG	no_sections < 1
引数	workspace	作業領域へのポインタへのポインタ
	no_sections	2次フィルタセクションの数 K
備考	本関数はリエントラントではありません。	

倍精度 IIR 作業領域解放***int FreeDfir (long **workspace, long no_sections)***

説明	InitDfir で割り付けられた作業領域メモリを解放します。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK	成功
	EDSP_BAD_ARG	no_section ≤ 2
引数	workspace	作業領域へのポインタへのポインタ
	no_sections	2次フィルタセクションの数 K
備考	本関数はリエントラントではありません。	

適応 FIR 作業領域解放***int FreeLms (short **workspace, long no_coefs)***

説明	InitLms で割り付けられた作業領域メモリを解放します。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK	成功
	EDSP_BAD_ARG	no_coefs < 1
引数	workspace	作業領域へのポインタへのポインタ
	no_coefs	係数の数 K
備考	本関数はリエントラントではありません。	

(7) 畳み込みと相関

(a) 関数一覧

関数名	説明
ConvComplete	2つの配列の完全な畳み込みを計算します。
ConvCyclic	2つの配列の周期的な畳み込みを計算します。
ConvPartial	2つの配列の部分的な畳み込みを計算します。
Correlate	2つの配列の相関を計算します。
CorrCyclic	2つの配列の周期的な相関を計算します。

これらの関数を使用する際は、2つの入力配列のうち1つはXメモリに、もう1つはYメモリに配置してください。出力配列はどのメモリに配置してもかまいません。

完全畳み込み

```
int ConvComplete (short output[ ], const short ip_x[ ], const short ip_y[ ],
long x_size, long y_size, int res_shift)
```

説明 2つの入力配列 x, y を完全に畳み込み、結果を出力配列 z に書き出します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ $x_size < 1$
・ $y_size < 1$
・ $res_shift < 0$
・ $res_shift > 25$

引数 output[] 出力 z
ip_x[] 入力 x
ip_y[] 入力 y
x_size ip_x のサイズ X
y_size ip_y のサイズ Y
res_shift 各出力に適用される右シフト

備考

$$z(m) = \left[\sum_{i=0}^{X-1} x(i) y(m-i) \right] \cdot 2^{-res_shift} \quad 0 \leq m < X+Y-1$$

入力配列外のデータは0として読み込まれます。
出力配列サイズは $X+Y-1$ 以上に設定してください。

周期的畳み込み

***int ConvCyclic (short output[], const short ip_x[], const short ip_y[],
long size, int res_shift)***

説明 2つの入力配列 x, y を周期的に畳み込み、結果を出力配列 z に書き出します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ size < 6
・ res_shift < 0
・ res_shift > 25

引数 output[] 出力 z
ip_x[] 入力 x
ip_y[] 入力 y
size 配列のサイズ N
res_shift 各出力に適用される右シフト

備考

$$z(m) = \left[\sum_{i=0}^{N-1} x(i) y(|m - i + N|_N) \right] \cdot 2^{-\text{res_shift}} \quad 0 \leq m < N$$

ここで、 $|i|_N$ は剰余 ($i \% N$) を意味します。

***int ConvPartial (short output[], const short ip_x[], const short ip_y[],
long x_size, long y_size, int res_shift)***

説明 本関数は 2 つの入力配列 x, y を畳み込み、結果を出力配列 z に書き出します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ $x_size < 5$
・ $y_size < 1$
・ $res_shift < 0$
・ $res_shift > 25$

引数 output[] 出力 z
ip_x[] 入力 x
ip_y[] 入力 y
x_size ip_x のサイズ X
y_size ip_y のサイズ Y
res_shift 各出力に適用される右シフト

備考 入力配列外のデータから引き出された出力は含まれていません。

$$z(m) = \left[\sum_{i=0}^{A-1} a(i) b(m + A - 1 - i) \right] \cdot 2^{-res_shift} \quad 0 \leq m \leq |A-B|$$

ただし、配列の個数は $a < b$ で、 A は a のサイズ、 B は b のサイズです。

出力配列サイズは $|X-Y|+1$ 以上に設定してください。

入力配列外のデータは 0 として読み込まれます。

相関

int Correlate (short output[], const short ip_x[], const short ip_y[], long x_size, long y_size, long no_corr, int x_is_larger, int res_shift)

説 明 2つの入力配列 x, y の相関を求め、結果を出力配列 z に書き出します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 • $x_size < 1$
 • $y_size < 1$
 • $no_corr < 1$
 • $res_shift < 0$
 • $res_shift > 25$
 • $x_is_larger \neq 0$ または 1

引 数 output[] 出力 z
 ip_x[] 入力 x
 ip_y[] 入力 y
 x_size ip_x のサイズ X
 y_size ip_y のサイズ Y
 no_corr 計算する相関の数 M
 x_is_larger $X=Y$ のときの配列指定
 res_shift 各出力に適用される右シフト

備 考 以下の式では配列の個数は $a > b$ で、 A は a のサイズとします。 $X=Y$ の場合は、 $x_is_larger=1$ とすると x を a とし、 $x_is_larger=0$ とすると x を b とします。

$$z(m) = \left[\sum_{i=0}^{A-1} a(i) b(i + m) \right] \cdot 2^{-res_shift} \quad 0 \leq m < M$$

$A < X + M$ となっても差し支えありません。この場合、入力配列外のデータは 0 を使用します。

$res_shift = 0$ は通常の整数計算に、 $res_shift = 15$ は小数計算に相当します。

int CorrCyclic (short output[], const short ip_x[], const short ip_y[], long size, int reverse, int res_shift)

説明 周期的に配列 x, y の相関を求め、結果を出力配列 z に書き出します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ size < 5
 ・ res_shift < 0
 ・ res_shift > 25
 ・ reverse ≠ 0 または 1

引数 output[] 出力 z
 ip_x[] 入力 x
 ip_y[] 入力 y
 size 配列のサイズ N
 reverse 反転フラグ
 res_shift 各出力に適用される右シフト

備考

$$z(m) = \left[\sum_{i=0}^{N-1} x(i) y(|i + m|_N) \right] \cdot 2^{-res_shift} \quad 0 \leq m < N$$

ここで、 $|i|_N$ は剰余 ($i \% N$) を意味します。reverse=1 の場合、出力のデータは反転され、実際の計算は以下のようになります。

$$z(m) = \left[\sum_{i=0}^{N-1} y(i) x(|i + m|_N) \right] \cdot 2^{-res_shift} \quad 0 \leq m < N$$

(8) その他

(a) 関数一覧

関数名	説明
Limit	H'8000 のデータを H'8001 に置き換えます。
CopyXtoY	配列を X メモリから Y メモリにコピーします。
CopyYtoX	配列を Y メモリから X メモリにコピーします。
CopyToX	配列を指定した場所から X メモリにコピーします。
CopyToY	配列を指定した場所から Y メモリにコピーします。
CopyFromX	配列を X メモリから指定した場所にコピーします。
CopyFromY	配列を Y メモリから指定した場所にコピーします。
GenGNoise	白色ガウス雑音を生成します。
MatrixMult	2 つのマトリックスの乗算をします。
VectorMult	2 つのデータの乗算をします。
MsPower	2 乗平均強度を求めます。
Mean	平均を求めます。
Variance	平均と偏差を求めます。
Maxl	整数配列の最大値を求めます。
Minl	整数配列の最小値を求めます。
Peakl	整数配列の最大絶対値を求めます。

H'8000 H'8001 置換***int Limit (short data_xy[], long no_elements, int data_is_x)***

説明 値が H'8000 の入力データを H'8001 に置き換えます。これにより、DSP 命令の固定小数点乗算の際にオーバーフローが発生しないようにします。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・no_elements < 1
 ・data_is_x ≠ 0 または 1

引数 data_xy[] データ配列
 no_elements データ数
 data_is_x データ配置指定

備考 この処理を行っても積和演算の加算でオーバーフローが発生する可能性はあります。data_is_x=1 のときはデータは X メモリに、data_is_x=0 のときはデータは Y メモリに配置してください。

Xメモリ Yメモリコピー***int CopyXtoY (short op_y[], const short ip_x[], long n)***

説明	配列を ip_x から op_y へコピーします。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK EDSP_BAD_ARG	成功 n < 6
引数	op_y[] ip_x[] n	出力配列 入力配列 データ数
備考	ip_x は Xメモリに、op_y は Yメモリに配置してください。	

Yメモリ Xメモリコピー***int CopyYtoX (short op_x[], const short ip_y[], long n)***

説明	配列を ip_y から op_x へコピーします。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK EDSP_BAD_ARG	成功 n < 1
引数	op_x[] ip_y[] n	出力配列 入力配列 データ数
備考	op_x は Xメモリに、ip_y は Yメモリに配置してください。	

X メモリへコピー***int CopyToX (short op_x[], const short input[], long n)***

説明	配列 input を op_x へコピーします。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK EDSP_BAD_ARG	成功 n < 1
引数	op_x[] input[] n	出力配列 入力配列 データ数
備考	op_x は X メモリに、input は任意のメモリに配置してください。	

Y メモリへコピー***int CopyToY (short op_y[], const short input[], long n)***

説明	配列 input を op_y へコピーします。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK EDSP_BAD_ARG	成功 n < 1
引数	op_y[] input[] n	出力配列 入力配列 データ数
備考	op_y は Y メモリに、input は任意のメモリに配置してください。	

Xメモリからコピー***int CopyFromX (short output[], const short ip_x[], long n)***

説明	配列 ip_x を output へコピーします。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK	成功
	EDSP_BAD_ARG	n < 1
引数	output[]	出力配列
	ip_x[]	入力配列
	n	データ数
備考	ip_x は Xメモリに、output は任意のメモリに配置してください。	

Yメモリからコピー***int CopyFromY (short output[], const short ip_y[], long n)***

説明	配列 ip_y を output へコピーします。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK	成功
	EDSP_BAD_ARG	n < 1
引数	output[]	出力配列
	ip_y[]	入力配列
	n	データ数
備考	ip_y は Yメモリに、output は任意のメモリに配置してください。	

白色ガウス雑音

int GenGNoise (short output[], long no_samples, float variance)

説明 平均が 0 で、ユーザが指定した偏差をもつ白色ガウス雑音を生成します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・no_samples < 1
・variance ≤ 0.0

引数 output[] 白色雑音データの出力
no_samples 出力データ数
variance ノイズ分布の偏差²

備考 出力データは 2 つ 1 組で生成されます。1 組の出力データを生成するために rand 関数を使用し、x の 2 乗合計が 1 未満になる組が求まるまで -1 ~ 1 の間で 1 組の乱数 r_1 、 r_2 を生成します。そして、1 組の出力データ r_1 、 r_2 が以下の式で計算されます。

$$o_1 = \sigma \gamma_1 \sqrt{-2 \ln(x)/x}$$

$$o_2 = \sigma \gamma_2 \sqrt{-2 \ln(x)/x}$$

データ数を奇数に設定した場合、最後の組の 2 番目のデータは破棄されます。

本関数が呼び出している標準ライブラリの rand 関数はリエントラントではないので、生成される乱数 r_1 、 r_2 の順番が常に同じになるとは限りません。しかし、生成される白色雑音 r_1 、 r_2 の特性に影響を及ぼすことはありません。

本関数は浮動小数点演算を使用しています。浮動小数点演算は処理速度が遅くなるので、本関数は評価用として使うことをおすすめします。

int MatrixMult (void *op_matrix, const void *ip_x, const void *ip_y, long m, long n, long p, int x_first, int res_shift)

説明 2つのマトリックス x, y の乗算を行い、結果を `op_matrix` に配置します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ m, n , または $p < 5$
 ・ $res_shift < 0$
 ・ $res_shift > 25$
 ・ $x_first \neq 0$ または 1

引数 `op_matrix` 出力の第一データへのポインタ
`ip_x` 入力 x の第一データへのポインタ
`ip_y` 入力 y の第一データへのポインタ
`m` マトリックス 1 の行数
`n` マトリックス 1 の列数、マトリックス 2 の行数
`p` マトリックス 2 の列数
`x_first` マトリックス乗算の順番指定
`res_shift` 各出力に適用される右シフト

備考 $x_first=1$ の場合、 $x \cdot y$ を計算します。このとき、`ip_x` は $m \times n$ 、`ip_y` は $n \times p$ 、`op_matrix` は $m \times p$ となります。

$x_first=0$ の場合、 $y \cdot x$ を計算します。このとき、`ip_y` は $m \times n$ 、`ip_x` は $n \times p$ 、`op_matrix` は $m \times p$ となります。

積和演算の結果は 39 ビットで保持されます。出力 $y(n)$ は `res_shift` ビット右シフトした結果の下位 16 ビットを取り出したものとなります。なお、オーバーフローしたときは正または負の最大値となります。

各マトリックスは通常の C 様式 (行優先順) で配置されます。

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \end{pmatrix}$$

任意の配列サイズを指定できるようにするために、配列パラメタは `void*` で指定します。これらのパラメタは `short` 変数を指すようにしてください。

入力配列 `ip_x, ip_y` と出力配列 `op_matrix` は別々に用意してください。

乗算

int VectorMult (short output[], const short ip_x[], const short ip_y[], long no_elements, int res_shift)

説明 ip_x, ip_y から 1 つずつデータを取り出して乗算を行い、結果を output に配置します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_elements < 4
 ・ res_shift < 0
 ・ res_shift > 16

引数 output[] 出力
 ip_x[] 入力 1
 ip_y[] 入力 2
 no_elements データ数
 res_shift 各出力に適用される右シフト

備考 出力は res_shift ビット右シフトした結果の下位 16 ビットを取り出したものとなります。
 なお、オーバーフローしたときは正または負の最大値となります。
 本関数はデータの乗算を行います。内積を計算する場合は m と p を 1 に設定して MatrixMult を使用してください。

平均 2 乗値

int MsPower (long *output, const short input[], long no_elements, int src_is_x)

説明 入力データの平均 2 乗値を求めます。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_elements < 6
 ・ src_is_x ≠ 0 または 1

引数 output 出力へのポインタ
 input[] 入力 x
 no_elements データ数 N
 src_is_x データ配置指定

備考 平均 2 乗値 = $\frac{1}{N} \sum_{i=0}^{N-1} x(i)^2$

除算結果は最も近い整数値に丸められます。

演算の結果は 63 ビットで保持されます。no_elements が 2³² 以上の場合、オーバーフローが発生することがあります。

src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。

int Mean (short *mean, const short input[], long no_elements, int src_is_x)

説明 入力データの平均値を求めます。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ no_elements < 6
・ src_is_x ≠ 0 または 1

引数 mean input の平均 \bar{x} へのポインタ
input[] 入力 x
no_elements データ数 N
src_is_x データ配置指定

備考
$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x(i)$$

除算結果は最も近い整数値に丸められます。

演算結果は 32 ビットで保持されます。no_elements が $2^{16}-1$ よりも大きい場合、オーバーフローが発生することがあります。

src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。

平均と偏差

***int Variance (long *variance, short *mean, const short input[],
long no_elements, int src_is_x)***

説 明 input の平均と偏差を求めます。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
 EDSP_BAD_ARG 以下のいずれかの場合です
 ・ no_elements < 4
 ・ src_is_x ≠ 0 または 1

引 数 variance 入力の偏差²へのポインタ
 mean データの平均 x へのポインタ
 input[] 入力 x
 no_elements データ数 N
 src_is_x データ配置指定

備 考
$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x(i)$$

$$\sigma^2 = \frac{1}{N} \sum_{i=0}^{N-1} x(i)^2 - \bar{x}^2$$

除算結果は最も近い整数値に丸められます。

\bar{x} は 32 ビットで保持されます。また、オーバーフローのチェックはしません。

no_elements が $2^{16}-1$ よりも大きい場合、オーバーフローが発生することがあります。

² は 63 ビットで保持されます。オーバーフローのチェックはしません。

src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。

最大値

int MaxI (short **max_ptr, short input[], long no_elements, int src_is_x)

説明 配列 input の最大値を検索して、そのアドレスを max_ptr に返します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ no_elements < 1
・ src_is_x ≠ 0 または 1

引数 max_ptr 最大データへのポインタへのポインタ
input[] 入力
no_elements データ数
src_is_x データ配置指定

備考 複数のデータが同じ最大値をもつ場合、input の先頭に最も近いデータのアドレスが返されます。
src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。

最小値

int MinI (short **min_ptr, short input[], long no_elements, int src_is_x)

説明 配列 input の最小値を検索して、そのアドレスを min_ptr に返します。

ヘッダ <ensigdsp.h>

リターン値 EDSP_OK 成功
EDSP_BAD_ARG 以下のいずれかの場合です
・ no_elements < 1
・ src_is_x ≠ 0 または 1

引数 min_ptr 最小データへのポインタへのポインタ
input[] 入力
no_elements データ数
src_is_x データ配置指定

備考 複数のデータが同じ最小値をもつ場合、input の先頭に最も近いデータのアドレスが返されます。
src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。

最大絶対値

int PeakI (short **peak_ptr, short input[], long no_elements, int src_is_x)

説明	配列 input の最大絶対値を検索して、そのアドレスを peak_ptr に返します。	
ヘッダ	<ensigdsp.h>	
リターン値	EDSP_OK EDSP_BAD_ARG	成功 以下のいずれかの場合です ・ no_elements < 1 ・ src_is_x ≠ 0 または 1
引数	peak_ptr input[] no_elements src_is_x	最大絶対値データへのポインタへのポインタ 入力 データ数 データ配置指定
備考	複数のデータが同じ最大絶対値をもつ場合、input の先頭に最も近いデータのアドレスが返されます。 src_is_x=1 のときはデータは X メモリに、src_is_x=0 のときはデータは Y メモリに配置してください。	

11. アセンブラ言語仕様

11.1 プログラムの要素

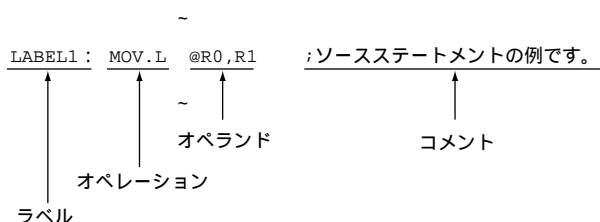
11.1.1 ソースステートメント

(1) ソースステートメントの構成

ソースステートメントの構成を以下に示します。

[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



(a) ラベル

ソースステートメントにつける名札としてシンボルまたはローカルシンボルを書きます。シンボルとはプログラマが定義する名前です。

(b) オペレーション

実行命令、DSP 命令、拡張命令、アセンブラ制御命令、各種制御文のニーモニック(名称)を書きます。実行命令、DSP 命令はマイコンの命令です。拡張命令はアセンブラが定義した命令で、実行命令と定数データ(リテラル)または複数の実行命令に展開される命令です。アセンブラ制御命令はアセンブラに指示を与える命令です。制御文には、ファイルインクルード機能、条件つきアセンブル機能、マクロ機能に関するものがあります

(c) 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。

(d) 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。

11. アセンブラ言語仕様

(2) ソースステートメントの書き方

ソースステートメントは ASCII 文字で記述します。ただし、文字列またはコメントの中にはかな・漢字(シフト JIS コード、EUC コード)、欧州コード文字を記述できます。基本的に 1 つのソースステートメントは 1 行に納まるように書いてください。1 行の最大長は 8,192 文字です。

(a) ラベルの書き方

ラベルは次のように書きます。

- 1 カラム目から書き始める。
または
- ラベル名の直後にコロンの(:)をつける。

良い例

```
LABEL1      ; 1 カラム目から書き始めています。  
LABEL2:    ; コロン(:)で終わっています。
```

悪い例

```
LABEL3      ; 1 カラム目から書き始めず、コロン(:)をつけてもいないので、  
            ; アセンブラはラベルと見なしません。
```

(b) オペレーションの書き方

オペレーションは次のように書きます。

- ラベルを記述していない場合：2 カラム目以降から書き始める。
- ラベルを記述している場合：ラベルとの間に 1 つ以上の空白またはタブを置いて書き始める。

コーディング例

```
                ADD          R0,R1          ; ラベルを記述していない場合です。  
LABEL1:        ADD          R1,R2          ; ラベルを記述してある場合です。
```

(c) オペランドの書き方

オペランドはオペレーションとの間に 1 つ以上の空白またはタブを置いて書き始めます。

コーディング例

```
ADD           R0,R1          ; ADD 命令のオペランドは 2 つです。  
SHAL         R1             ; SHAL 命令のオペランドは 1 つです。
```

(d) コメントの書き方

セミコロン(;)の後に続けて書きます。アセンブラはセミコロンから行末までをコメントと見なしません。

コーディング例

```
ADD          R0,R1          ; R1 に R0 を加えます。
```

(3) 複数行にわたるソースステートメントの書き方

次のようなときにはプログラムを見やすくするため、1つのソースステートメントを複数の行に分けて書くことができます。

- ソースステートメントが長くなる場合
- オペランドの1つ1つにコメントをつけたい場合

複数行にわたるソースステートメントは次の(a)~(c)の手順で記述してください。

- オペランドとオペランドを区切るカンマ(,)を切れ目として改行します。
- すぐ次の行の1カラム目にプラス(+を書きます。
- そのプラスの後ろにソースステートメントの続きを書きます。

プラスの後ろに空白またはタブを入れても構いません。各行の最後にコメントを書くこともできます。

コーディング例 1

```

        .DATA.L          H'FFFF0000,
+
        H'FF00FF00,
+
        H'FFFFFFFF
; 1つのソースステートメントを3行にわたって書いた例です。

```

コーディング例 2

```

        .DATA.L          H'FFFF0000,          ; 初期値 1
+
        H'FF00FF00,          ; 初期値 2
+
        H'FFFFFFFF          ; 初期値 3
; オペランド1つ1つに、コメントをつけた例です。

```

11.1.2 予約語

予約語は特別な意味を持つ語としてアセンブラが用意している名前です。予約語にはレジスタ名、演算子、ロケーションカウンタがあります。レジスタ名はCPU種別ごとに異なりますので各CPUの「プログラミングマニュアル」を参照してください。予約語はシンボルとしては使用できません。

- レジスタ名
R0 ~ R15、FR0 ~ FR15、DR0 ~ DR14(偶数番号のみ)、XD0 ~ XD14(偶数番号のみ)、FV0 ~ FV12(4の倍数の番号のみ)、R0_BANK ~ R7_BANK、SP^{*}、SR、GBR、VBR、MACH、MACL、PR、PC、SSR、SPC、FPUL、FPSCR、MOD、RE、RS、DSR、A0、A0G、A1、A1G、M0、M1、X0、X1、Y0、Y1、XMTRX、DBR、SGR
- 演算子
STARTOF、SIZEOF、HIGH、LOW、HWORD、LWORD、\$EVEN、\$ODD、\$EVEN2、\$ODD2
- ロケーションカウンタ
\$

【注】* R15 と SP は同じレジスタを表します。

11.1.3 シンボル

(1) シンボルの役割

シンボルはプログラマが定義する名前であり、次の役割を果たします。

- アドレスシンボル：データの格納場所、分岐先などのアドレスを表します。
- 定数シンボル：定数を表します。
- レジスタ別名：汎用レジスタおよび浮動小数点レジスタを表します。
- セクション名：セクションの名前を表します。

シンボルの使用例を以下に示します。

コーディング例 1

```
~  
BRA SUB1 ; BRA は分岐命令です。  
~ ; SUB1 は分岐先のアドレスシンボルを表します。  
SUB1:  
~
```

コーディング例 2

```
~  
MAX: .EQU 100 ; .EQU はシンボルに値を設定するアセンブラ制御命令です。  
MOV.B #MAX,R0 ; MAX は値 100 を表します。  
~
```

コーディング例 3

```
~  
MIN: .REG R0 ; .REG はレジスタ別名を定義するアセンブラ制御命令です。  
MOV.B #100,MIN ; MIN は R0 の別名になります。  
~
```

コーディング例 4

```
~  
.SECTION CD, CODE, ALIGN=4 ; .SECTION はセクションを宣言するアセンブラ制御命令です。  
~ ; CD はこのセクションの名前となります。  
~
```

(2) シンボルの名付け方

(a) シンボルに使用できる文字

次の ASCII 文字を使用できます。

- 英大文字、英小文字(A~Z、a~z)
- 数字(0~9)
- アンダースコア(_)
- ドル(\$)

アセンブラはシンボル中の英大文字と英小文字を区別します。

(b) 先頭の文字

次のいずれかに限ります。

- 英大文字、英小文字(A~Z、a~z)
- アンダースコア(_)
- ドル(\$)

【注】ドル(\$)1文字はロケーションカウンタを表す予約語です。

(c) 最大文字数

とくに制限はありません。

(d) シンボルとして使用できない名前

予約語はシンボルとして使用できません。また、以下に示すような名前はアセンブラが内部シンボルとして使用します。プログラマが同じ名前を使うことはできません。

`_$nmmnn` (n は数字(0~9)です)

【注】* 内部シンボルとはアセンブラの内部処理のため必要なシンボルです。内部シンボルはアセンブルリストやオブジェクトモジュールには出力されません。

(e) シンボルの定義と参照

シンボルはラベルとして記述することにより定義されます。参照するときはオペランドに記述します。例外として、`.SECTION` と `.MACRO` では定義するシンボルをオペランドに記述します。また、`.MACRO` で定義したシンボル(マクロ名)はオペレーションに記述して参照します(マクロコール)。

シンボルを参照する場合、定義よりも参照が先に現れることがあります。このような参照を前方参照と呼びます。通常はこのような参照も問題なく可能ですが、一部で許されないことがありますので注意が必要です。

複数のソースファイルでプログラムを構成する場合、ファイルをまたがるシンボル参照が必要になります。定義したシンボルを他のソースファイルから参照できるようにすることを外部定義と呼びます。逆に、他のソースファイルで定義したシンボルを参照することを外部参照と呼びます。外部定義は `.EXPORT`、`.GLOBAL` で宣言します。外部参照は `.IMPORT`、`.GLOBAL` で宣言します。外部参照も前方参照と同様、許されないことがありますので注意が必要です。

11.1.4 定数

(1) 整数定数

整数定数は基数をつけて表現します。基数とはその整数定数が何進数であることを示す記述です。

- 2進数 …… 基数 B' と 2進表現の数値で記述
- 8進数 …… 基数 Q' と 8進表現の数値で記述
- 10進数 …… 基数 D' と 10進表現の数値で記述
- 16進数 …… 基数 H' と 16進表現の数値で記述

アセンブラは基数の英大文字と英小文字を区別しません。基数と数値は間を空けずに続けて書いてください。基数は省略しても構いません。基数のない整数定数は(通常は)10進数として扱われます(.RADIXによって何進数にするかを指定できます)。

コーディング例

```
.DATA.B B'10001000 ;
.DATA.B Q'210 ; これらのソースステートメントは、全く同じ内容
.DATA.B D'136 ; を表しています。
.DATA.B H'88 ;
```

【補足】

- B' : BINARY(2進)の意味です。
- Q' : OCTAL(8進)の意味です。Oは数字のゼロと紛らわしいのでQを使います。
- D' : DECIMAL(10進)の意味です。
- H' : HEXADECIMAL(16進)の意味です。

(2) 文字定数

文字定数は文字コードを値とする定数です。4バイト以内の文字をダブルコーテーション(")で囲んで記述してください。ASCII文字、シフトJISコードもしくはEUCコードのかな・漢字、欧州コード文字を記述することができます。ASCII文字はH'09(タブ)、H'20(空白)~H'7E(~)が使用できます。ダブルコーテーション自身をデータとして使う場合は2つ続けて書いてください。かな・漢字を記述した場合、シフトJISコードのときはsjisオプション、EUCコードのときはeucオプションを、欧州コード文字を記述した場合は、latin1オプションを指定してください。なお、シフトJISコード、EUCコード、欧州コードを混在して使うことはできません。

コーディング例 1

```
.DATA.L "ABC" ;.DATA.L H'00414243 同じ意味です。
.DATA.W "AB" ;.DATA.W H'4142 同じ意味です。
.DATA.B "A" ;.DATA.B H'41 同じ意味です。
; AのASCIIコード… H'41
; BのASCIIコード… H'42
; CのASCIIコード… H'43
```

コーディング例 2

```
.DATA.B "" ;ダブルコーテーション 1文字の文字定数です。
.DATA.L "漢字" ;漢字
```


(3) 浮動小数点定数

浮動小数点定数は浮動小数点定数確保のアセンブラ制御命令で指定することができます。

(a) 浮動小数点定数の書き方

浮動小数点定数の表記には 10 進表記と 16 進表記の 2 種類があります。

• 10 進表記

$$F' [\{ \pm \}] \left\{ \begin{array}{l} n [\cdot [m]] \\ .m \end{array} \right\} [t [[\{ \pm \}] xx]]$$

F' : 10 進浮動小数点定数であることを示します。省略はできません。
 $[\{ \pm \}]$: n には整数部を 10 進で指定します。 m には小数部を 10 進で指定します。整数部と小数部の値は、どちらか一方を省略できます。また、 \pm を省略すると $+$ を仮定します。
 t : 精度のコードを指定します。次の 2 種類があります。
 ・ S 単精度
 ・ D 倍精度
 省略するとアセンブラ制御命令のオペレーションサイズに従います。
 $[\{ \pm \}]xx$: 指数部(10 のべき乗)の値を 10 進で指定します。
 省略すると 0 乗を仮定します。また、 \pm を省略すると $+$ を仮定します。

例

$F' 0.5S - 2 = 0.5 \times 10^{-2} = 0.005 = H' 3BA3D70A$
 $F' .123D3 = 0.123 \times 10^3 = 123 = H' 405EC00000000000$

• 16 進表記

 $H' xxxx [\cdot t]$

H' : 16 進であることを示します。省略はできません。
 $xxxx$: 浮動小数点定数のビットパターンを 16 進で指定します。データ長より指定が短い場合は右づめに確保し、左側は必要分の 0 をつめます。長い場合は指定の右側の有効長分のデータを確保します。
 t : 精度のコードを指定します。
 次の 2 種類があります。
 ・ S 単精度
 ・ D 倍精度
 省略するとアセンブラ制御命令のオペレーションサイズに従います。

この形式は、精度幅の 0 や無限大表示など 10 進表記の書き方では表記しにくいデータを記述するもので、確保すべき浮動小数点定数のビットパターンを直接指定します。

例

$H' 0123456789ABCDEF.S$ $H' 89ABCDEF$
 $H' FFFF.D$ $H' 000000000000FFFF$

11. アセンブラ言語仕様

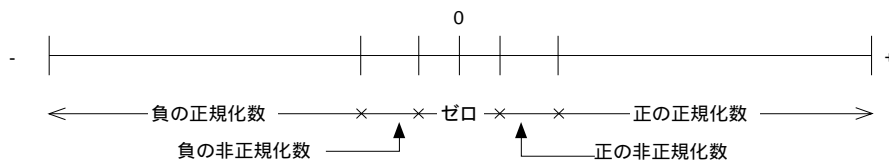
(b) 浮動小数点値の範囲

表 11.1 に浮動小数点値のデータタイプを示します。

表 11.1 浮動小数点値のデータタイプ

データタイプ	内容
正規化数(Normalized Number)	絶対値がアンダフロー境界以上でオーバフロー境界以下の値
非正規化数(Denormalized Number)	絶対値が 0 より大きくアンダフロー境界未満の値
ゼロ	絶対値が 0 の値
無限大	絶対値がオーバフローより大きい値
非数(Not a Number : NAN)	数値でない値 sNAN(signaling NAN)と qNAN(quiet NAN)があります

これらを数直線上に表すと次のようになります。ただし、非数は数値として扱わないため数直線上には表せません。



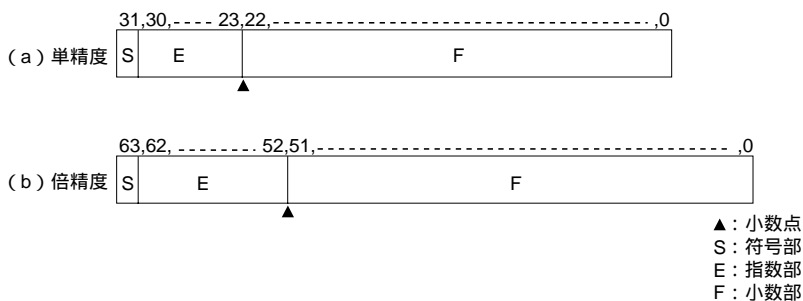
次にアセンブラで記述できる数値範囲を表 11.2 に示します。

表 11.2 データ形式と数値範囲(絶対値)

データ形式		単精度	倍精度
正規化数	最大値	3.40×10^{38}	1.79×10^{308}
	最小値	1.18×10^{-38}	2.23×10^{-308}
非正規化数	最大値	1.17×10^{-38}	2.22×10^{-308}
	最小値	1.40×10^{-45}	4.94×10^{-324}

(c) 浮動小数点データ形式

FPU の浮動小数点データ形式を示します。



- ・符号部(S)
数値の符号を表現します。0で正数、1で負数です。
- ・指数部(E)
指数を表現します。データフォーマット中の指数部の値からある一定のバイアス値(bias)を引いた値が実際の指数になります。
- ・小数部(F)
小数部は各ビットごとに重みをもっており、先頭ビットから $2^{-1}, 2^{-2}, \dots, 2^{-n}$ (nは小数部のビット長)に対応しています。

次にデータ形式のサイズを表 11.3 に示します。

表 11.3 データ形式のサイズ

パラメータ	単精度	倍精度
ビット長	32 ビット	64 ビット
符号ビット(S)	1 ビット	1 ビット
指数部(E)	8 ビット	11 ビット
小数部(F)	23 ビット	52 ビット
指数のバイアス(bias)	127	1023

浮動小数点の数値は表 11.3 の記号を用いると次のように表わせます。

$$2^{E-\text{bias}} \cdot (-1)^S \cdot \begin{cases} (1.F) & \text{: 正規化数} \\ (0.F) & \text{: 非正規化数} \end{cases}$$

$$(1.F) = 1 + b_0 \times 2^{-1} + b_1 \times 2^{-2} + \dots + b_{n-1} \times 2^{-n} \quad \begin{array}{l} b: \text{小数部のビット位置} \\ n: \text{小数部のビット長} \end{array}$$

$$(0.F) = b_0 \times 2^{-1} + b_1 \times 2^{-2} + \dots + b_{n-1} \times 2^{-n}$$

次にデータタイプごとの浮動小数点表現を示すと表 11.4 のようになります。非数は数値でないため表わせません。

表 11.4 データタイプの浮動小数点表現

データタイプ	単精度	倍精度
正規化数	$(-1)^S \cdot 2^{E-127} \cdot (1.F)$	$(-1)^S \cdot 2^{E-1023} \cdot (1.F)$
非正規化数	$(-1)^S \cdot 2^{-126} \cdot (0.F)$	$(-1)^S \cdot 2^{-1022} \cdot (0.F)$
ゼロ	$(-1)^S \cdot 0$	
無限大	$(-1)^S \cdot \infty$	
非数	quiet NAN, signaling NAN	

(d) 浮動小数点定数の有効数字

浮動小数点定数確保の制御命令において、本アセンブラがオブジェクトコードを生成するとき、以下2通りの丸め方をサポートし、有効数字を設定します。

- (a) Round to Nearest even(RN) : オブジェクトコード最下位ビットを最近値に丸めます。
最近値が2つのとき、最下位ビットがゼロになるように丸めます。
- (b) Round to Zero(RZ) : オブジェクトコード最下位ビットを切り捨てて0にします。

例

```
.FDATA.S F' 1S-1 のオブジェクトコード
RN の場合 : H' 3DCCCCCD
RZ の場合 : H' 3DCCCCC
```

(e) 非正規化数の扱い

非正規化数の扱いはCPU種別により異なります。非正規化数を扱わないCPUの場合、非正規化数の範囲の値はウォーニング841とし、0のオブジェクトコードを出力します。非正規化数を扱うCPUの場合、非正規化数の範囲の値はウォーニング842とし、非正規化数のオブジェクトコードを出力します。非正規化数の扱いはオプションで変更することができます。

例

- 非正規化数を扱わないCPUの場合
.FDATA.S F' 1S-40 : ウォーニング841、オブジェクトコードH' 00000000
- 非正規化数を扱うCPUの場合
.FDATA.S F' 1S-40 : ウォーニング842、オブジェクトコードH' 000116C2

(4) 固定小数点定数

固定小数点定数は固定小数点定数確保のアセンブラ制御命令のオペランドで指定することができます。

(a) 固定小数点定数の書き方

固定小数点定数は-1.0~1.0までの範囲の実数データが扱え、10進数で記述します。固定小数点定数にはワードサイズとロングワードサイズの2種類があります。

- ワードサイズ固定小数点定数

2バイトの符号付き整数で-1.0~1.0の範囲の実数を表現します。2バイトの符号付き整数の値 x (-32,768 $\leq x \leq$ 32,767)が表現する実数値は $x/32768$ となります。

例

固定小数点定数	ワードデータの表現
- 1.0	H'8000
- 0.5	H'C000
0.0	H'0000
0.5	H'4000
1.0	H'7FFF

- ロングワードサイズ固定小数点定数

4 バイトの符号付き整数で-1.0 ~ 1.0 の範囲の実数を表現します。4 バイトの符号付き整数の値 x (-2,147,483,648 \times 2,147,483,647) が表現する実数値は $x/2147483648$ となります。

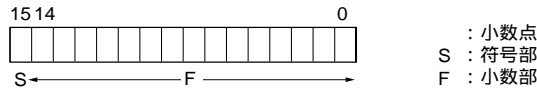
例

固定小数点定数	ロングワードデータの表現
- 1.0	H'80000000
- 0.5	H'C0000000
0.0	H'00000000
0.5	H'40000000
1.0	H'7FFFFFFF

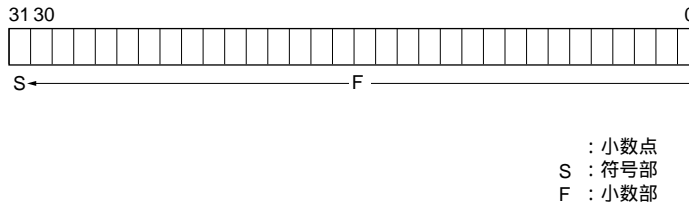
(b) 固定小数点データ形式

固定小数点定数のデータ形式はワードサイズの場合、符号 1 ビットと数値 15 ビット、ロングワードサイズの場合、符号 1 ビットと数値 31 ビットからなるもので、小数点は常に符号部の右側にあるものと考えます。

ワードサイズ



ロングワードサイズ



- 符号部(S)

数値の符号を表します。0で正数、1で負数です。

- 小数部(F)

小数部は各ビットごとに重みをもっており、先頭ビットから $2^{-1}, 2^{-2}, \dots, 2^{-31}$ に対応します。

(c) 固定小数点定数の有効数字

ロングワードサイズの場合、31 ビットで表現できる値は 10 進で 9 桁となりますが、アセンブラは有効数字を 10 進 10 桁の小数で扱い、32 ビット目を RN(絶対値が 0 に近い方に丸める)で丸め、計算結果の上位 31 ビットを固定小数点データとします。

【注】 実際の固定小数点データの範囲は-1.0 ~ 0.999999999 ですが、1.0 は 0.999999999 と仮定し、H'7FFFFFFF とします。

11.1.5 ロケーションカウンタ

ロケーションカウンタはオブジェクトコード(実行命令やデータをコンピュータが理解できる形式に変換したコード)を配置するアドレス(ロケーション)を指し示します。ロケーションカウンタ値はオブジェクトコードの出力に応じて自動的に変化します。また、アセンブラ制御命令により意図的にロケーションカウンタ値を変えることもできます。

コーディング例

```

~
.ORG      H'00001000      ;ロケーションカウンタに H'00001000 を設定しています。
.DATA.W   H'FF           ;このアセンブラ制御命令のオブジェクトコードは長さ 2 バイトです。
                        ;ロケーションカウンタ値は H'00001002 に変わります。
.DATA.W   H'F0           ;このアセンブラ制御命令のオブジェクトコードは長さ 2 バイトです。
                        ;ロケーションカウンタ値は H'00001004 に変わります。
.DATA.W   H'10           ;このアセンブラ制御命令のオブジェクトコードは長さ 2 バイトです。
                        ;ロケーションカウンタ値は H'00001006 に変わります。
.ALIGN    4              ;ロケーションカウンタ値を 4 の倍数に補正しています。
                        ;ロケーションカウンタ値は H'00001008 に変わります。
.DATA.L   H'FFFFFFFF     ;このアセンブラ制御命令のオブジェクトコードは長さ 4 バイトです。
                        ;ロケーションカウンタ値は H'0000100C に変わります。
; .ORG はロケーションカウンタ値を設定するアセンブラ制御命令です。
; .ALIGN はロケーションカウンタ値を補正するアセンブラ制御命令です。
; .DATA はデータをメモリ上に確保するアセンブラ制御命令です。
; .W はデータをワード (=2 バイト) 単位で扱うとの指定です。
; .L はデータをロングワード (=4 バイト) 単位で扱うとの指定です。
~

```

ロケーションカウンタはドル(\$)で参照できます。

コーディング例

```

LABEL1:   .EQU $         ;LABEL1 というシンボルにこの時点でのロケーション
                        ;カウンタ値を設定しています。
; .EQU はシンボルに値を設定するアセンブラ制御命令です。

```

11.1.6 式

式は定数やシンボルと演算子を組み合わせて演算結果を求めるものであり、実行命令やアセンブラ制御命令のオペランドに使用します。

(1) 式の要素

式は項、演算子、カッコから構成されます。

(a) 項

項には次のものがあります。

- 定数
- ロケーションカウンタ(\$)
- シンボル(レジスタ別名を除く)
- 上記の項と演算子による演算結果

単独の項も式の種類です。

(b) 演算子

表 11.5 に演算子の一覧を示します。

表 11.5 演算子一覧

演算区分	演算子	演算内容	書き方
算術演算	+	単項プラス	+ 項
	-	単項マイナス	- 項
	+	加算	項 1 + 項 2
	-	減算	項 1 - 項 2
	*	乗算	項 1 * 項 2
	/	除算	項 1 / 項 2
論理演算	~	単項否定	~ 項
	&	論理積	項 1 & 項 2
		論理和	項 1 項 2
	~	排他的論理和	項 1 ~ 項 2
シフト演算	<<	算術左シフト	項 1 << 項 2
	>>	算術右シフト	項 1 >> 項 2
セクション集合演算	STARTOF	セクション集合の先頭アドレスを求める	STARTOF セクション名
	SIZEOF	セクション集合のサイズをバイト単位で求める	SIZEOF セクション名
偶奇演算	\$EVEN	2の倍数の時1、そうでない時0	\$EVEN シンボル
	\$ODD	2の倍数の時0、そうでない時1	\$ODD シンボル
	\$EVEN2	4の倍数の時1、そうでない時0	\$EVEN2 シンボル
	\$ODD2	4の倍数の時0、そうでない時1	\$ODD2 シンボル
抽出演算	HIGH	上位バイト抽出	HIGH 項
	LOW	下位バイト抽出	LOW 項
	HWORD	上位ワード抽出	HWORD 項
	LWORD	下位ワード抽出	LWORD 項

11. アセンブラ言語仕様

(c) カッコ

丸カッコ()によって演算の優先順位を変更できます。

(2) 演算の順序

1つの式の中に複数の演算が含まれる場合、演算子の優先順位とカッコ指定によって演算を処理する順序が決まります。アセンブラは次の規則にしたがって演算を処理します。

- 規則 1
カッコで括られた演算から処理する。
カッコが多重になっているときはより内側のカッコで括られた演算を優先する。
- 規則 2
演算子の優先順位が高いものから処理する。
- 規則 3
演算子の優先順位が同じであるときは演算子の結合規則の向きに処理する。

表 11.6 に演算子の優先順位と結合規則を示します。

表 11.6 演算子の優先順位と結合規則

優先順位	演算子	結合規則
1 (高)	+ - ~ STARTOF SIZEOF \$EVEN \$ODD \$EVEN2 \$ODD2 HIGH LOW HWORD LWORD *	右から左の順に演算を処理する。
2	* /	左から右の順に演算を処理する。
3	+ -	左から右の順に演算を処理する。
4	<< >>	左から右の順に演算を処理する。
5	&	左から右の順に演算を処理する。
6 (低)	~	左から右の順に演算を処理する。

【注】*優先順位 1 の演算子は単項演算子です。

式の記述例を以下に示します。

コーディング例 1

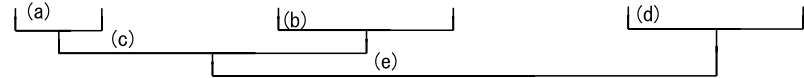
$$1 + (2 - (3 + (4 - 5)))$$

アセンブラは、(a)～(d)の順に計算します。

(a)の結果 -1	} 最終的な結果は、1になります。
(b)の結果 2	
(c)の結果 0	
(d)の結果 1	

コーディング例 2

- H' FFFFFFFF1 + H' 000000F0 * H' 00000010 | H' 000000F0 & H' 0000FFFF

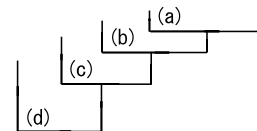


アセンブラは、(a)～(e)の順に計算します。

(a)の結果	H' 0000000F	} 最終的な結果は、H' 0000FFFになります。
(b)の結果	H' 0000F00	
(c)の結果	H' 0000F0F	
(d)の結果	H' 00000F0	
(e)の結果	H' 0000FFF	

コーディング例 3

- ~ - ~ H' 0000000F



アセンブラは、(a)～(d)の順に計算します。

(a)の結果	H' FFFFFFF0	} 最終的な結果は、H' 00000011になります。
(b)の結果	H' 00000010	
(c)の結果	H' FFFFFFFF	
(d)の結果	H' 00000011	

(3) 演算の詳細

(a) STARTOF 演算

指定したセクションが最適化リンケージエディタで連結された後のセクション先頭アドレスを求めます。

(b) SIZEOF 演算

指定したセクションが最適化リンケージエディタで連結された後のセクションサイズを求めます。

11. アセンブラ言語仕様

コーディング例

```

        .CPU          SH1
        .SECTION      INIT_RAM, DATA, ALIGN=4
        .RES.B        H'100
        .SECTION      INIT_DATA, DATA, ALIGN=4
INIT_BGN .DATA.L      (STARTOF INIT_RAM)           ; [1]
INIT_END .DATA.L      (STARTOF INIT_RAM) + (SIZEOF INIT_RAM) ; [2]
;
;
        .SECTION      MAIN, CODE, ALIGN=4
INITIAL:
        MOV.L        DATA1, R6
        MOV          #0, R5
        MOV.L        DATA1+4, R3
        BRA          LOOP2
        MOV.L        @R3, R4

LOOP1:
        MOV.L        R5, @R4
        ADD          #4, R4

LOOP2:
        MOV.L        @R6, R3
        CMP/HI       R3, R4
        BF           LOOP1
        RTS
        NOP

DATA1:
        .DATA.L      INIT_END
        .DATA.L      INIT_BGN
        .END

```

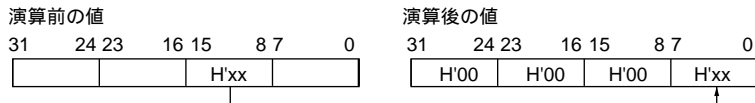
セクション(INIT_RAM)の
データ領域をゼロで初期化
します。

[1]: セクション(INIT_RAM)の先頭アドレスを求めます。

[2]: セクション(INIT_RAM)の最終アドレスを求めます。

(c) HIGH 演算

4 バイト値の下位 2 バイトの上位バイトを抽出します。



コーディング例

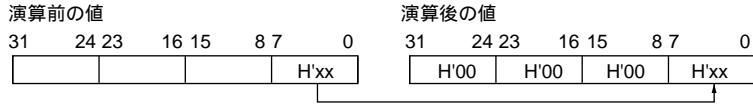
```

LABEL   .EQU          H'00007FFF
        .DATA          HIGH LABEL           ; メモリ上に整数データ
                                                ; H'0000007F を確保します。

```

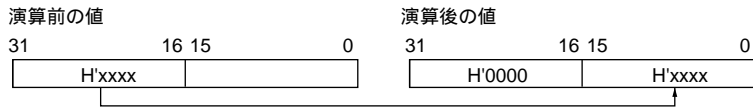
(d) LOW 演算

4 バイト値の最下位バイトを抽出します。



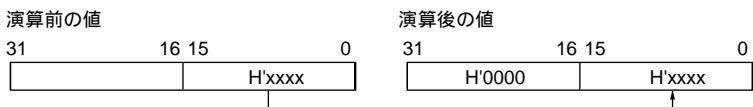
(e) HWORD 演算

4 バイト値の上位 2 バイトを抽出します。



(f) LWORD 演算

4 バイト値の下位 2 バイトを抽出します。



(g) 偶奇演算

アドレスシンボルの値が 2 の倍数または 4 の倍数かを判定します。
表 11.7 に偶奇演算子の演算内容を示します。

表 11.7 偶奇演算子の演算内容

演算子	演算内容
\$EVEN	2 の倍数の時 1、そうでない時 0
\$ODD	2 の倍数の時 0、そうでない時 1
\$EVEN2	4 の倍数の時 1、そうでない時 0
\$ODD2	4 の倍数の時 0、そうでない時 1

コーディング例

\$ODD2 演算子を使用して現在のプログラムカウンタを求める例です。

```

LAB:
    MOVA    @(0,PC),R0
    ADD     #-4+2*$ODD2 LAB,R1      ; $ODD2 は LAB が
                                    ; 4 の倍数ならば 0 とし、
                                    ; 4 の倍数でないなら 1 とします。
    
```

11. アセンブラ言語仕様

(4) 式に関する注意事項

(a) 内部処理

アセンブラは式の値を 32 ビット符号付きで処理します。

コーディング例

```
~H'F0
```

アセンブラは H'F0 を H'000000F0 と解釈します。したがって、~H'F0 は H'FFFFFF0F であり、H'0000000F ではありません。

(b) 算術演算

値が確定しなければならない箇所では乗算、除算で相対アドレスまたは外部参照シンボルを項とすることはできません。また、除算で 0 を除数とすることはできません。

コーディング例

```
.IMPORT      SYM
.DATA        SYM/10      ; 正常となる
.ORG        SYM/10      ; エラーとなる
```

(c) 論理演算

論理演算で相対アドレスまたは外部参照シンボルを項とすることはできません。

11.1.7 文字列

文字列は一連の文字をデータとして考えます。文字列には次の ASCII 文字を使用できます。

```
ASCIIコード  ┌ H'09(タブ)
               └ H'20(空白)~H'7E(~)
```

文字列中の 1 文字はその文字の ASCII コードを値としてもつバイトサイズのデータを表します。また、シフト JIS コードもしくは EUC コードのかな・漢字、欧州コード文字も記述することができます。文字としてかな・漢字を記述した場合、シフト JIS コードのときは sjis オプション、EUC コードのときは euc オプションを指定してください。指定しない場合はホストマシンに依存する日本語コードとします。欧州コード文字を記述した場合、latin1 オプションを指定してください。

文字列は文字の並びをダブルコーテーション(")で囲んで記述してください。データを表す文字としてダブルコーテーションを使う場合はダブルコーテーションを 2 つ続けて書いてください。

コーディング例

```
.SDATA          "Hello!"          ;文字列データ Hello!を確保しています。
.SDATA          "アセンブラ"      ;文字列データアセンブラを確保しています。
.SDATA          "" "Hello!" ""     ;文字列データ"Hello!"を確保しています。
```

; .SDATA は文字列データをメモリ上に確保するアセンブラ制御命令です。

【注】文字定数と文字列の違い

文字定数は数値です。データのサイズは 1 バイト、2 バイト、4 バイトのいずれかになります。文字列は数値として扱えません。データのサイズは 1 バイト以上 255 バイト以下です。

11.1.8 ローカルラベル

(1) ローカルラベルの役割

ローカルラベルはアドレスシンボル間で局所的に有効なラベルです。ローカルラベルは有効範囲外の他のシンボルと衝突することがありませんので、他のシンボル名を意識せずに局所的な制御ができます。ローカルラベルは通常のアドレスシンボルと同様にラベルに記述することによって定義でき、オペランド内で参照できます。

なお、ローカルラベルはデバッグ時に参照できないほか、次の位置に指定できません。

- マクロ名
- セクション名
- オブジェクトモジュール名
- .ASSGINA、.ASSIGNC、.EQU、.ASSIGN、.REG、.FREG のラベル
- .EXPORT、.IMPORT、.GLOBAL のオペランド

ローカルラベルの使用例を以下に示します。

コーディング例

```

LABEL1:                                     ; ローカルブロック 1 の開始
?0001:
    ~
    CMP/EQ          R1,R2
    BT              ?0002          ; ローカルブロック 1 の?0002 に分岐します。
    BRA            ?0001          ; ローカルブロック 1 の?0001 に分岐します。
?0002:
    ~
LABEL2:                                     ; ローカルブロック 2 の開始
?0001:
    ~
    CMP/GE          R1,R2
    BT              ?0002          ; ローカルブロック 2 の?0002 に分岐します。
    BRA            ?0001          ; ローカルブロック 2 の?0001 に分岐します。
?0002:
LABEL3:                                     ; ローカルブロック 3 の開始

```

(2) ローカルラベルの名付け方

(a) 文字の先頭

ローカルラベルは先頭が"?"で始まる文字列です。

(b) ローカルラベルに使用できる文字

ローカルラベルは先頭以外の文字が以下のASCII文字からなる文字列です。

- ・ 英大文字、英小文字(A~Z、a~z)
- ・ 数字(0~9)
- ・ アンダースコア(_)
- ・ ドル(\$)

アセンブラは英大文字と英小文字を区別しています。

(c) 最大文字数

2文字以上16文字以内です。17文字以上記述するとローカルシンボルとして扱われません。

(3) ローカルラベルの有効範囲

ローカルラベルの有効範囲をローカルブロックといいます。ローカルブロックの区切りはアドレスシンボルまたは.SECTIONです。このローカルブロック内で定義されたローカルラベルは当該ローカルブロック内で参照できます。異なるローカルブロックに属するローカルラベルは、同じ名前であっても別のラベルと解釈し、エラーとはなりません。

【注】 .ASSIGNA、ASSIGNC、.EQU、.ASSIGN、.REG、.FREG で定義したアドレスシンボルは有効範囲の区切りとはみなしません。

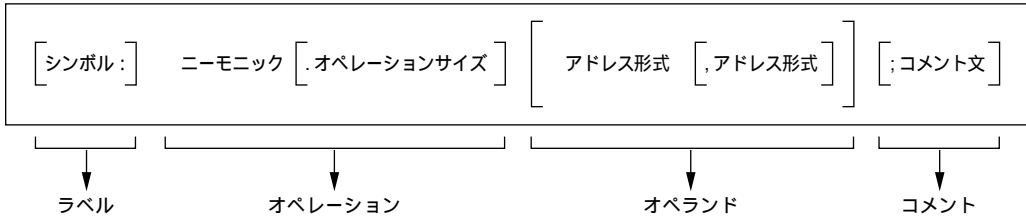
11.2 実行命令

11.2.1 実行命令の概要

実行命令はマイコンの命令です。

マイコンはメモリ上にある実行命令のオブジェクトコードを解読して実行します。

実行命令のソースステートメントは基本的に次のように記述します。



本節ではニーモニック、オペレーションサイズ、アドレス形式について説明します。

(1) ニーモニック

ニーモニックは実行命令を表します。処理内容を連想させる英文字の名前がニーモニックとして用意されています。

アセンブラはニーモニック中の英大文字と英小文字を区別しません。

(2) オペレーションサイズ

オペレーションサイズはデータを操作する単位です。

指定できるオペレーションサイズは実行命令ごとに異なります。

アセンブラはオペレーションサイズの英大文字と英小文字を区別しません。

指定内容	データのサイズ
B	バイト
W	ワード(2バイト)
L	ロングワード(4バイト)
S	単精度(4バイト)
D	倍精度(8バイト)

(3) アドレス形式

アクセスの対象となるデータ領域や分岐先アドレスなどを表します。

指定できるアドレス形式は実行命令ごとに異なります。

表 11.8 にアドレス形式の一覧を示します。

表 11.8 アドレス形式一覧

アドレス形式	名称	解説
Rn	レジスタ直接	レジスタ上の領域です。
@Rn	レジスタ間接	メモリ上の領域です。Rnの値が領域の先頭アドレスを表します。
@Rn+	ポストインクリメントレジスタ間接	メモリ上の領域です。インクリメント ¹ する前のRnの値が領域の先頭アドレスを表します。 マイコンはRnを先に参照し、後からインクリメントします。
@-Rn	プリデクリメントレジスタ間接	メモリ上の領域です。デクリメント ² した後のRnの値が、領域の先頭アドレスを表します。マイコンはRnを先にデクリメントし、後から参照します。
@(disp,Rn)	ディスプレースメント付きレジスタ間接 ³	メモリ上の領域です。領域の先頭アドレスは、Rnの値+ディスプレースメント(disp)です。 Rnの内容は変わりません。
@(R0,Rn)	インデックス付きレジスタ間接	メモリ上の領域です。領域の先頭アドレスは、Rnの値+R0の値です。 RnおよびR0の内容は変わりません。
@(disp,GBR)	ディスプレースメント付きGBR間接	メモリ上の領域です。領域の先頭アドレスはGBRの値+ディスプレースメント(disp)です。GBRの内容は変わりません。
@(R0,GBR)	インデックス付きGBR間接	メモリ上の領域です。領域の先頭アドレスはGBRの値+R0の値です。 GBRおよびR0の内容は変わりません。
@(disp,PC)	ディスプレースメント付きPC相対	メモリ上の領域です。領域の先頭アドレスはPCの値+ディスプレースメント(disp)です。
symbol	シンボル指定によるPC相対	【分岐命令のオペランドである場合】 symbolは分岐先のアドレスを表します。アセンブラはsymbolとPCの値からディスプレースメント(disp)を逆算します。 disp=symbol-PCです。 【データ転送命令のオペランドである場合】 メモリ上の領域です。symbolは領域の先頭アドレスを表します。アセンブラはsymbolとPCの値からディスプレースメント(disp)を逆算します。disp=symbol-PCです。 【RS,REレジスタを指定する命令(LDRS,LDRE)のオペランドである場合】 メモリ上の領域です。symbolは領域の先頭アドレスを表します。アセンブラはsymbolとPCの値からディスプレースメント(disp)を逆算します。disp=symbol-PCです。
#imm	イミディエイト	定数を表します。

【注】 *1 インクリメント

オペレーションサイズがバイトのとき1、ワード(2バイト)のとき2、ロングワード(4バイト)のとき4を加えることです。

*2 デクリメント

オペレーションサイズがバイトのとき1、ワード(2バイト)のとき2、ロングワード(4バイト)のとき4を減じることです。

*3 ディスプレースメント

2点間の距離です。本アセンブリ言語ではバイト単位で表現します。

ディスプレースメントとして許される値は、オペランドのアドレス形式やオペレーションサイズによって異なります。

表 11.9 ディスプレースメントとして許される値

アドレス形式	ディスプレースメント〔単位はバイト()内は 10 進表現〕
@(disp,Rn)	オペレーションサイズがバイト(B)のとき H'00000000 ~ H'0000000F(0 ~ 15) オペレーションサイズがワード(W)のとき H'00000000 ~ H'0000001E(0 ~ 30) オペレーションサイズがロングワード(L)のとき H'00000000 ~ H'0000003C(0 ~ 60)
@(disp,GBR)	オペレーションサイズがバイト(B)のとき H'00000000 ~ H'000000FF(0 ~ 255) オペレーションサイズがワード(W)のとき H'00000000 ~ H'000001FE(0 ~ 510) オペレーションサイズがロングワード(L)のとき H'00000000 ~ H'000003FC(0 ~ 1020)
@(disp,PC)	【データ転送命令のオペランドである場合】 オペレーションサイズがワード(W)のとき H'00000000 ~ H'000001FE(0 ~ 510) オペレーションサイズがロングワード(L)のとき H'00000000 ~ H'000003FC(0 ~ 1020) 【RS、RE レジスタを設定する命令(LDRS,LDRE)のオペランドである場合】 H'FFFFFF00 ~ H'000000FE(-256 ~ 254)
symbol	【分岐命令のオペランドである場合】 条件付きの分岐命令(BT,BF,BF/S,BT/S)のオペランドであるとき H'00000000 ~ H'000000FF(0 ~ 255) H'FFFFFF00 ~ H'FFFFFFF(-256 ~ -1) 無条件の分岐命令(BRA、BSR)のオペランドであるとき H'00000000 ~ H'00000FFF(0 ~ 4095) H'FFFFFF00 ~ H'FFFFFFF(-4096 ~ -1) 【データ転送命令のオペランドである場合】 オペレーションサイズがワード(W)のとき H'00000000 ~ H'000001FE(0 ~ 510) オペレーションサイズがロングワード(L)のとき H'00000000 ~ H'000003FC(0 ~ 1020) 【RS、RE レジスタを設定する命令(LDRS,LDRE)のオペランドである場合】 H'FFFFFF00 ~ H'000000FE(-256 ~ 254)

11. アセンブラ言語仕様

イミディエイトとして許される値は実行命令によって異なります。

表 11.10 イミディエイトとして許される値

実行命令	イミディエイト
TST,AND,OR,XOR	H'00000000 ~ H'000000FF(0 ~ 255)
MOV	H'00000000 ~ H'000000FF(0 ~ 255) H'FFFFFF80 ~ H'FFFFFF(-128 ~ -1) ¹⁾
ADD,CMP/EQ	H'00000000 ~ H'000000FF(0 ~ 255) H'FFFFFF80 ~ H'FFFFFF(-128 ~ -1) ¹⁾
TRAPA	H'00000000 ~ H'000000FF(0 ~ 255)
SETRC	H'00000001 ~ H'000000FF(1 ~ 255) ²⁾

【注】 *1 H'FFFFFF80 ~ H'FFFFFF の範囲を正の 10 進数で記述しても構いません。

*2 SETRC 命令のイミディエイトデータに 0 を設定した場合、ウォーニング 835 とし、オブジェクトコードには 0 を設定します。この場合、リポートする範囲は 1 回実行されます。また、SETRC 命令のイミディエイト値に外部参照シンボルを指定した場合、最適化リンケージエディタは H'00000000 ~ H'000000FF(0 ~ 255)の範囲のチェックとなります。

【注】 アセンブラは条件に応じてディスプレースメントを補正します。

条件	補正内容
オペレーションサイズがワードで ディスプレースメントが 2 の倍数でない	ディスプレースメントの下位 1 ビット を切り捨て、2 の倍数に補正
オペレーションサイズがロングワードで ディスプレースメントが 4 の倍数でない	ディスプレースメントの下位 2 ビット を切り捨て、4 の倍数に補正
分岐命令で ディスプレースメントが 2 の倍数でない	ディスプレースメントの下位 1 ビット を切り捨て、2 の倍数に補正

オペランドとして@(disp,Rn)、@(disp,GBR)、@(disp,PC)を記述する場合にはアセンブラによるディスプレースメントの補正を考慮してください。

コーディング例

```
MOV.L @(63,PC),R0
```

アセンブラはディスプレースメントを 63 から 60 に補正して MOV.L @(60,PC), R0 と同じオブジェクトコードを生成します。また、ウォーニング 870 を通知します。

11.2.2 実行命令に関する注意事項

(1) オペレーションサイズに関する注意

ニーモニックとアドレス形式の組み合わせにより指定できるオペレーションサイズが異なります。

(a) SH-1 の実行命令とオペレーションサイズの組み合わせ

表 11.11 に SH-1 の実行命令とオペレーションサイズの組み合わせを示します。

表 11.11 実行命令とオペレーションサイズの組み合わせ(その 1)

ニーモニック	データ転送命令 アドレス形式	オペレーションサイズ			
		B	W	L	省略時
MOV	#imm,Rn				B ^{*1}
MOV	@(disp,PC),Rn	x			L
MOV	symbol,Rn	x			L
MOV	Rn,Rm	x	x		L
MOV	Rn,@Rm				L
MOV	@Rn,Rm				L
MOV	Rn,@-Rm				L
MOV	@Rn+,Rm				L
MOV	R0,@(disp,Rn)				L
MOV	Rn,@(disp,Rm)	x	x		L ^{*2}
MOV	@(disp,Rn),R0				L
MOV	@(disp,Rn),Rm	x	x		L ^{*3}
MOV	Rn,@(R0,Rm)				L
MOV	@(R0,Rn),Rm				L
MOV	R0,@(disp,GBR)				L
MOV	@(disp,GBR),R0				L
MOVA	#imm,R0	x	x		L
MOVA	@(disp,PC),R0	x	x		L
MOVA	symbol,R0	x	x		L
MOVT	Rn	x	x		L
SWAP	Rn,Rm			x	W
XTRCT	Rn,Rm	x	x		L

【注】 *1 サイズ選択モードの場合、アセンブラが imm の値によりオペレーションサイズを決めます。

*2 この場合の Rn は R1 ~ R15

*3 この場合の Rm は R1 ~ R15

11. アセンブラ言語仕様

表 11.11 実行命令とオペレーションサイズの組み合わせ(その2)

ニーモニック	算術演算命令 アドレス形式	オペレーションサイズ			省略時
		B	W	L	
ADD	Rn,Rm	x	x		L
ADD	#imm,Rn	x	x		L
ADDC	Rn,Rm	x	x		L
ADDV	Rn,Rm	x	x		L
CMP/EQ	#imm,R0	x	x		L
CMP/EQ	Rn,Rm	x	x		L
CMP/HS	Rn,Rm	x	x		L
CMP/GE	Rn,Rm	x	x		L
CMP/HI	Rn,Rm	x	x		L
CMP/GT	Rn,Rm	x	x		L
CMP/PZ	Rn	x	x		L
CMP/PL	Rn	x	x		L
CMP/STR	Rn,Rm	x	x		L
DIV1	Rn,Rm	x	x		L
DIV0S	Rn,Rm	x	x		L
DIV0U	(オペランドなし)	x	x	x	-
EXTS	Rn,Rm			x	W
EXTU	Rn,Rm			x	W
MAC	@Rn+,@Rm+	x		x	W
MULS	Rn,Rm	x			L
MULU	Rn,Rm	x			L
NEG	Rn,Rm	x	x		L
NEGC	Rn,Rm	x	x		L
SUB	Rn,Rm	x	x		L
SUBC	Rn,Rm	x	x		L
SUBV	Rn,Rm	x	x		L

【注】*WとLは同じオブジェクトコードとなります。

表 11.11 実行命令とオペレーションサイズの組み合わせ(その 3)

ニーモニック	論理演算命令		オペレーションサイズ		
	アドレス形式	B	W	L	省略時
AND	Rn,Rm	x	x		L
AND	#imm,R0	x	x		L
AND	#imm,@(R0,GBR)		x	x	B
NOT	Rn,Rm	x	x		L
OR	Rn,Rm	x	x		L
OR	#imm,R0	x	x		L
OR	#imm,@(R0,GBR)		x	x	B
TAS	@Rn		x	x	B
TST	Rn,Rm	x	x		L
TST	#imm,R0	x	x		L
TST	#imm,@(R0,GBR)		x	x	B
XOR	Rn,Rm	x	x		L
XOR	#imm,R0	x	x		L
XOR	#imm,@(R0,GBR)		x	x	B

表 11.11 実行命令とオペレーションサイズの組み合わせ(その 4)

ニーモニック	シフト命令		オペレーションサイズ		
	アドレス形式	B	W	L	省略時
ROTL	Rn	x	x		L
ROTR	Rn	x	x		L
ROTCL	Rn	x	x		L
ROTCR	Rn	x	x		L
SHAL	Rn	x	x		L
SHAR	Rn	x	x		L
SHLL	Rn	x	x		L
SHLR	Rn	x	x		L
SHLL2	Rn	x	x		L
SHLR2	Rn	x	x		L
SHLL8	Rn	x	x		L
SHLR8	Rn	x	x		L
SHLL16	Rn	x	x		L
SHLR16	Rn	x	x		L

11. アセンブラ言語仕様

表 11.11 実行命令とオペレーションサイズの組み合わせ(その 5)

ニーモニック	分岐命令	オペレーションサイズ			省略時
	アドレス形式	B	W	L	
BF	symbol	x	x	x	-
BT	symbol	x	x	x	-
BRA	symbol	x	x	x	-
BSR	symbol	x	x	x	-
JMP	@Rn	x	x	x	-
JSR	@Rn	x	x	x	-
RTS	(オペランドなし)	x	x	x	-

表 11.11 実行命令とオペレーションサイズの組み合わせ(その 6)

ニーモニック	システム制御命令	オペレーションサイズ			省略時
	アドレス形式	B	W	L	
CLRT	(オペランドなし)	x	x	x	-
CLRMAC	(オペランドなし)	x	x	x	-
LDC	Rn,SR	x	x		L
LDC	Rn,GBR	x	x		L
LDC	Rn,VBR	x	x		L
LDC	@Rn+,SR	x	x		L
LDC	@Rn+,GBR	x	x		L
LDC	@Rn+,VBR	x	x		L
LDS	Rn,MACH	x	x		L
LDS	Rn,MACL	x	x		L
LDS	Rn,PR	x	x		L
LDS	@Rn+,MACH	x	x		L
LDS	@Rn+,MACL	x	x		L
LDS	@Rn+,PR	x	x		L
NOP	(オペランドなし)	x	x	x	-
RTE	(オペランドなし)	x	x	x	-
SETT	(オペランドなし)	x	x	x	-
SLEEP	(オペランドなし)	x	x	x	-
STC	SR,Rn	x	x		L
STC	GBR,Rn	x	x		L
STC	VBR,Rn	x	x		L
STC	SR,@-Rn	x	x		L
STC	GBR,@-Rn	x	x		L
STC	VBR,@-Rn	x	x		L
STS	MACH,Rn	x	x		L
STS	MACL,Rn	x	x		L
STS	PR,Rn	x	x		L
STS	MACH,@-Rn	x	x		L
STS	MACL,@-Rn	x	x		L
STS	PR,@-Rn	x	x		L
TRAPA	#imm	x	x		L

【記号の意味】

Rm	: 汎用レジスタ(R0 ~ R15)
Rn	: 汎用レジスタ(R0 ~ R15)
R0	: 汎用レジスタ(R0 固定)
SR	: ステータスレジスタ
GBR	: グローバル・ベースレジスタ
VBR	: ベクタ・ベースレジスタ
MACH	: 積和レジスタ(上位)
MACL	: 積和レジスタ(下位)
PR	: プロシージャレジスタ
PC	: プログラムカウンタ
Imm	: イミディエイト
disp	: ディスプレースメント
symbol	: シンボル
B	: バイト
W	: ワード(2 バイト)
L	: ロングワード(4 バイト)
	: 指定は有効
x	: 指定は無効、オペレーションサイズの指定を省略したのと同じ結果になる : アセンブラは拡張命令として解釈する

(b) SH-2 の実行命令とオペレーションサイズの組み合わせ

表 11.12 に SH-1 に対して SH-2 で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 11.12 実行命令とオペレーションサイズの組み合わせ(その 1)

ニーモニック	算術演算命令 アドレス形式	オペレーションサイズ			省略時
		B	W	L	
MAC	@Rn+, @Rm+	x			W
MUL	Rn, Rm	x	x		L
DMULS	Rn, Rm	x	x		L
DMULU	Rn, Rm	x	x		L
DT	Rn	x	x	x	-

表 11.12 実行命令とオペレーションサイズの組み合わせ(その 2)

ニーモニック	分岐命令 アドレス形式	オペレーションサイズ			省略時
		B	W	L	
BF/S	symbol	x	x	x	-
BT/S	symbol	x	x	x	-
BRAF	Rn	x	x	x	-
BSRF	Rn	x	x	x	-

11. アセンブラ言語仕様

(c) SH-2E の実行命令とオペレーションサイズの組み合わせ

表 11.13 に SH-2 に対して SH-2E で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 11.13 実行命令とオペレーションサイズの組み合わせ(その 1)

データ転送命令		オペレーションサイズ				
ニーモニック	アドレス形式	B	W	L	S	省略時
FLDI0	FRn	x	x	x		S
FLDI1	FRn	x	x	x		S
FMOV	@Rm,FRn	x	x	x		S
FMOV	FRn,@Rm	x	x	x		S
FMOV	@Rm+,FRn	x	x	x		S
FMOV	FRn,@-Rm	x	x	x		S
FMOV	@(R0,Rm),FRn	x	x	x		S
FMOV	FRm,@(R0,Rm)	x	x	x		S
FMOV	FRm,FRn	x	x	x		S

表 11.13 実行命令とオペレーションサイズの組み合わせ(その 2)

算術演算命令		オペレーションサイズ				
ニーモニック	アドレス形式	B	W	L	S	省略時
FABS	FRn	x	x	x		S
FADD	FRm,FRn	x	x	x		S
FCMP/EQ	FRm,FRn	x	x	x		S
FCMP/GT	FRm,FRn	x	x	x		S
FDIV	FRm,FRn	x	x	x		S
FMAC	FR0,FRm,FRn	x	x	x		S
FMUL	FRm,FRn	x	x	x		S
FNEG	FRn	x	x	x		S
FSUB	FRm,FRn	x	x	x		S

表 11.13 実行命令とオペレーションサイズの組み合わせ(その3)

システム制御命令		オペレーションサイズ				省略時
ニーモニック	アドレス形式	B	W	L	S	
FLDS	FRm,FPUL	x	x	x		S
FLOAT	FPUL,FRn	x	x	x		S
FSTS	FPUL,FRn	x	x	x		S
FTRC	FRm,FPUL	x	x	x		S
LDS	Rm,FPUL	x	x		x	L
LDS	@Rm+,FPUL	x	x		x	L
LDS	Rm,FPSCR	x	x		x	L
LDS	@Rm+,FPSCR	x	x		x	L
STS	FPUL,Rn	x	x		x	L
STS	FPUL,@-Rn	x	x		x	L
STS	FPSCR,Rn	x	x		x	L
STS	FPSCR,@-Rn	x	x		x	L

【記号の意味】

- FRm : 浮動小数点レジスタ
- FRn : 浮動小数点レジスタ
- FR0 : 浮動小数点レジスタ(FR0 固定)
- FPUL : 浮動小数点通信レジスタ
- FPSCR : 浮動小数点ステータス/コントロールレジスタ
- S : 単精度(4 バイト)

(d) SH-3の実行命令とオペレーションサイズの組み合わせ

表 11.14 に SH-2 に対して SH-3 で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 11.14 実行命令とオペレーションサイズの組み合わせ(その1)

データ転送命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略時
PREF	@Rn	x	x	x	-

表 11.14 実行命令とオペレーションサイズの組み合わせ(その2)

シフト演算命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略時
SHAD	Rn, Rm	x	x		L
SHLD	Rn, Rm	x	x		L

11. アセンブラ言語仕様

表 11.14 実行命令とオペレーションサイズの組み合わせ(その3)

システム制御命令		オペレーションサイズ			
二ーモニック	アドレス形式	B	W	L	省略時
CLRS	(オペランドなし)	x	x	x	-
SETS	(オペランドなし)	x	x	x	-
LDC	Rm, SSR	x	x		L
LDC	Rm, SPC	x	x		L
LDC	Rm, Rn_BANK	x	x		L
LDC	@Rm+, SSR	x	x		L
LDC	@Rm+, SPC	x	x		L
LDC	@Rm+, Rn_BANK	x	x		L
STC	SSR, Rn	x	x		L
STC	SPC, Rn	x	x		L
STC	Rm_BANK, Rn	x	x		L
STC	SSR, @-Rn	x	x		L
STC	SPC, @-Rm	x	x		L
STC	Rm_BANK, @-Rn	x	x		L
LDTLB	(オペランドなし)	x	x	x	-

【記号の意味】

- Rn_BANK : バンク汎用レジスタ
- SSR : セーブ・ステータスレジスタ
- SPC : セーブ・プログラムカウンタ

(e) SH-4 の実行命令とオペレーションサイズの組み合わせ

表 11.15 に SH-3 に対して SH-4 で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 11.15 実行命令とオペレーションサイズの組み合わせ(その 1)

データ転送命令		オペレーションサイズ					省略時
ニーモニック	アドレス形式	B	W	L	S	D	
FLDI0	FRn	x	x	x		x	S
FLDI1	FRn	x	x	x		x	S
FMOV	FRm,FRn	x	x	x		x	S
FMOV	FRn,@Rm	x	x	x		x	S
FMOV	FRn,@-Rm	x	x	x		x	S
FMOV	FRn,@(R0,Rm)	x	x	x		x	S
FMOV	@Rm,FRn	x	x	x		x	S
FMOV	@Rm+,FRn	x	x	x		x	S
FMOV	@(R0,Rm),FRn	x	x	x		x	S
FMOV	DRm,DRn	x	x	x	x		D
FMOV	DRm,@Rn	x	x	x	x		D
FMOV	DRm@-Rn	x	x	x	x		D
FMOV	DRm,@(R0,Rn)	x	x	x	x		D
FMOV	@Rm,DRn	x	x	x	x		D
FMOV	@Rm+,DRn	x	x	x	x		D
FMOV	@(R0,Rm),DRn	x	x	x	x		D
FMOV	DRm,XDn	x	x	x	x		D
FMOV	XDm,DRn	x	x	x	x		D
FMOV	XDm,XDn	x	x	x	x		D
FMOV	XDm,@Rn	x	x	x	x		D
FMOV	XDm,@-Rn	x	x	x	x		D
FMOV	XDm,@(R0,Rn)	x	x	x	x		D
FMOV	@Rm,XDn	x	x	x	x		D
FMOV	@Rm+,XDn	x	x	x	x		D
FMOV	@(R0,Rm),XDn	x	x	x	x		D

11. アセンブラ言語仕様

表 11.15 実行命令とオペレーションサイズの組み合わせ(その2)

算術演算命令		オペレーションサイズ					省略時
ニーモニック	アドレス形式	B	W	L	S	D	
FABS	FRn	x	x	x		x	S
FABS	DRn	x	x	x	x		D
FADD	FRm,FRn	x	x	x		x	S
FADD	DRm,DRn	x	x	x	x		D
FCMP/EQ	FRm,FRn	x	x	x		x	S
FCMP/EQ	DRm,DRn	x	x	x	x		D
FCMP/GT	FRm,FRn	x	x	x		x	S
FCMP/GT	DRm,DRn	x	x	x	x		D
FDIV	FRm,FRn	x	x	x		x	S
FDIV	DRm,DRn	x	x	x	x		D
FIPR	FVm,FVn	x	x	x		x	S
FMAC	FR0,FRm,FRn	x	x	x		x	S
FMUL	FRm,FRn	x	x	x		x	S
FMUL	DRm,DRn	x	x	x	x		D
FNEG	FRn	x	x	x		x	S
FNEG	DRn	x	x	x	x		D
FSQRT	FRn	x	x	x		x	S
FSQRT	DRn	x	x	x	x		D
FSUB	FRm,FRn	x	x	x		x	S
FSUB	DRm,DRn	x	x	x	x		D
FTRV	XMTRX,FVn	x	x	x		x	S

表 11.15 実行命令とオペレーションサイズの組み合わせ(その3)

システム制御命令		オペレーションサイズ					省略時
ニーモニック	アドレス形式	B	W	L	S	D	
FCNVDS	DRm,FPUL	x	x	x	x		D
FCNVSD	FPUL,DRn	x	x	x	x		D
FLDS	FRm,FPUL	x	x	x		x	S
FLOAT	FPUL,FRn	x	x	x		x	S
FLOAT	FPUL,DRn	x	x	x	x		D
FRCHG	(オペランドなし)	x	x	x	x	x	
FSCHG	(オペランドなし)	x	x	x	x	x	
FSTS	FPUL,FRn	x	x	x		x	S
FTRC	FRm,FPUL	x	x	x		x	S
FTRC	DRm,FPUL	x	x	x	x		D
LDC	Rm,DBR	x	x		x	x	L
LDC	@Rm+,DBR	x	x		x	x	L
LDS	Rm,FPUL	x	x		x	x	L
LDS	@Rm+,FPUL	x	x		x	x	L
LDS	Rm,FPSCR	x	x		x	x	L
LDS	@Rm+,FPSCR	x	x		x	x	L
OCBI	@Rn	x	x	x	x	x	
OCBP	@Rn	x	x	x	x	x	
OCBWB	@Rn	x	x	x	x	x	
STC	DBR,Rn	x	x		x	x	L
STC	DBR,@-Rn	x	x		x	x	L
STC	SGR,Rn	x	x		x	x	L
STC	SGR,@-Rn	x	x		x	x	L
STS	FPUL,Rm	x	x		x	x	L
STS	FPUL,@-Rm	x	x		x	x	L
STS	FPSCR,Rm	x	x		x	x	L
STS	FPSCR,@-Rm	x	x		x	x	L

【記号の意味】

- DRm : 倍精度浮動小数点レジスタ
- DRn : 倍精度浮動小数点レジスタ
- XDm : 拡張倍精度浮動小数点レジスタ
- XDn : 拡張倍精度浮動小数点レジスタ
- FVm : 単精度浮動小数点ベクトルレジスタ
- FVn : 単精度浮動小数点ベクトルレジスタ
- XMTRX : 単精度浮動小数点拡張レジスタ行列
- DBR : デバッグベクタベースレジスタ
- SGR : 退避ジェネラルレジスタ 15
- D : 倍精度(8バイト)

11. アセンブラ言語仕様

(f) SH2-DSP、SH3-DSPの実行命令とオペレーションサイズの組み合わせ

表 11.16 に SH-2 に対して SH2-DSP、SH-3 に対して SH3-DSP で追加された実行命令とオペレーションサイズの組み合わせを示します。

表 11.16 実行命令とオペレーションサイズの組み合わせ(その 1)

リピート制御命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略時
LDRS	@(disp, PC)	x	x		L
LDRS	symbol	x	x		L
LDRE	@(disp, PC)	x	x		L
LDRE	symbol	x	x		L
SETRC	Rn	x	x	x	-
SETRC	#imm	x	x	x	-

表 11.16 実行命令とオペレーションサイズの組み合わせ(その 2)

システム制御命令		オペレーションサイズ			
ニーモニック	アドレス形式	B	W	L	省略時
LDC	Rn, MOD	x	x		L
LDC	Rn, RS	x	x		L
LDC	Rn, RE	x	x		L
LDC	@Rn+, MOD	x	x		L
LDC	@Rn+, RS	x	x		L
LDC	@Rn+, RE	x	x		L
LDS	Rn, DSR	x	x		L
LDS	Rn, A0	x	x		L
LDS	@Rn+, DSR	x	x		L
LDS	@Rn+, A0	x	x		L
STC	MOD, Rn	x	x		L
STC	RS, Rn	x	x		L
STC	RE, Rn	x	x		L
STC	MOD, @-Rn	x	x		L
STC	RS, @-Rn	x	x		L
STC	RE, @-Rn	x	x		L
STS	DSR, Rn	x	x		L
STS	A0, Rn	x	x		L
STS	DSR, @-Rn	x	x		L
STS	A0, @-Rn	x	x		L

【記号の意味】

- MOD : モジュロレジスタ
- RS : リピート・スタートレジスタ
- RE : リピート・エンドレジスタ
- DSR : DSP・ステータスレジスタ
- A0 : DSP・データレジスタ(A0 以外に A1、X0、X1、Y0、Y1 が指定できます)

(2) 遅延分岐命令に関する注意

無条件の分岐命令は遅延分岐命令です。

マイコンは遅延分岐命令の実行に先だってディレイスロット命令(メモリ上で遅延分岐命令の直後に位置する命令)を実行します。

ディレイスロット命令が不当なものである場合、アセンブラはエラー-150または151を通知します。遅延分岐命令とディレイスロット命令との関係を表 11.17 に示します。

表 11.17 遅延分岐命令とディレイスロット命令の関係

ディレイスロット命令	遅延分岐命令									
	BF/S	BT/S	BRAF	BSRF	BRA	BSR	JMP	JSR	RTS	RTE
BF	x	x	x	x	x	x	x	x	x	x
BT	x	x	x	x	x	x	x	x	x	x
BF/S	x	x	x	x	x	x	x	x	x	x
BT/S	x	x	x	x	x	x	x	x	x	x
BRAF	x	x	x	x	x	x	x	x	x	x
BSRF	x	x	x	x	x	x	x	x	x	x
BRA	x	x	x	x	x	x	x	x	x	x
BSR	x	x	x	x	x	x	x	x	x	x
JMP	x	x	x	x	x	x	x	x	x	x
JSR	x	x	x	x	x	x	x	x	x	x
RTS	x	x	x	x	x	x	x	x	x	x
RTE	x	x	x	x	x	x	x	x	x	x
TRAPA	x	x	x	x	x	x	x	x	x	x
LDC	Rn,SR	*1	*1	*1	*1	*1	*1	*1	*1	*1
	@Rn+,SR	*1	*1	*1	*1	*1	*1	*1	*1	*1
MOV	@(disp,PC),Rn	x	x	x	x	x	x	x	x	x
	symbol,Rn	x	x	x	x	x	x	x	x	x
MOVA	@(disp,PC),R0	x	x	x	x	x	x	x	x	x
	symbol,R0	x	x	x	x	x	x	x	x	x
拡張	MOV.L #imm,Rn	x	x	x	x	x	x	x	x	x
命令	MOV.W #imm,Rn	x	x	x	x	x	x	x	x	x
	MOVA #imm,R0	x	x	x	x	x	x	x	x	x
上記以外の命令										

【記号の意味】

- : 正常 アセンブラは指定どおりにオブジェクトコードを生成
- x : エラー-150または151
ディレイスロット命令が不当
アセンブラはNOP命令のオブジェクトコード(H'0009)を生成
- *1 : CPU種別がSH-1、SH-2、SH-2E、SH2-DSPのとき正常
上記以外のときエラー-150または151

11. アセンブラ言語仕様

【注】遅延分岐命令とディレイスロット命令が別々のセクションに属する場合、アセンブラはディレイスロット命令の正当性をチェックしません。

(3) アドレス計算に関する注意

オペランドのアドレス形式がディスプレイメント付き PC 相対@(disp,PC)である場合、PC の値を考慮してプログラミングする必要があります。

PC の値がどうなるかは状況によって異なります。

(a) 通常

PC の値は(実行中の命令の先頭アドレス+4)バイトです。

例

絶対アドレス H'00001000 の MOV 命令を実行している最中と考えてください。

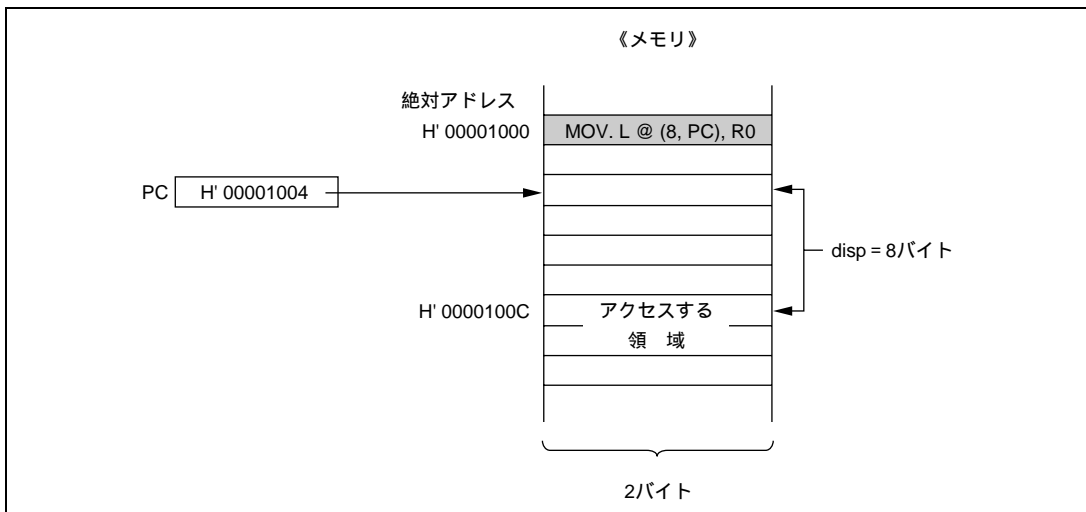


図 11.1 アドレス計算の例(通常)

(b) ディレイスロット命令を実行中

PC の値は(遅延分岐命令による分岐先アドレス+2)バイトとなります。

例

絶対アドレス H'00001000 の MOV 命令を実行している最中と考えてください

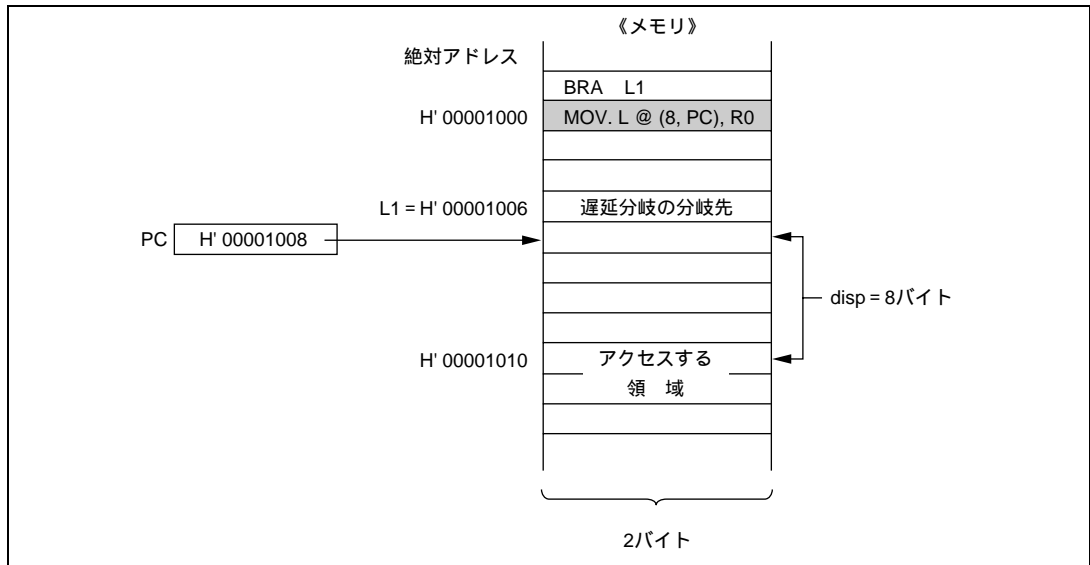


図 11.2 アドレス計算の例(分岐によって PC が変化する場合)

【補足】 オペランドがシンボル指定による PC 相対(symbol)である場合には、アセンブラは PC の値を考慮した上でディスプレースメントを逆算し、オブジェクトコードを生成します。

11. アセンブラ言語仕様

(c) MOV.L @(disp,PC),Rn、および MOVA @(disp,PC),R0 のどちらかを実行中

マイコンはPCの値が4の倍数でないとき、下位2ビットを切り捨てて4の倍数に補正し、アドレスを計算します。

例1 マイコンがPCを補正する場合

H'00001002番地のMOV命令を実行している最中と考えてください。

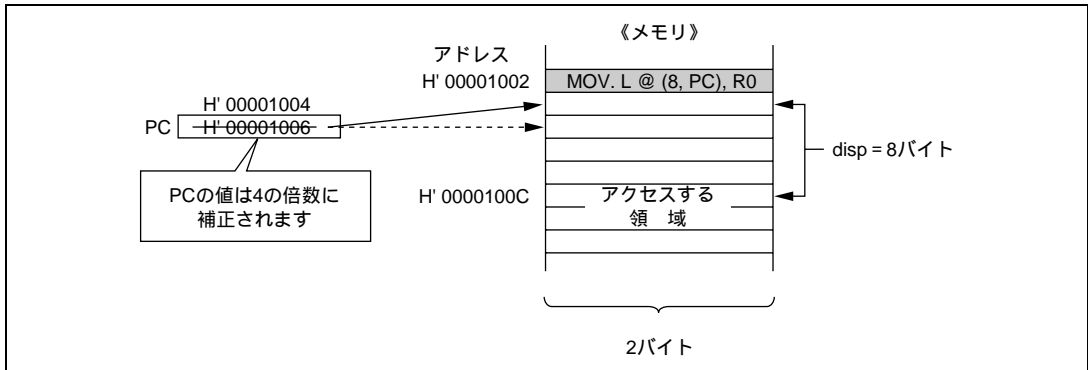


図 11.3 アドレス計算の例(マイコンがPCを補正する場合)

例2 マイコンがPCを補正しない場合

H'00001000番地のMOV命令を実行している最中と考えてください。

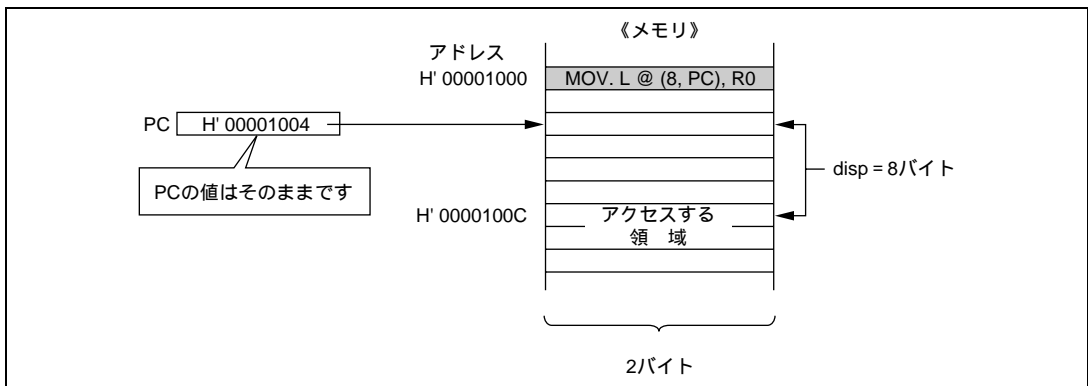


図 11.4 アドレス計算の例(マイコンがPCを補正しない場合)

【補足】 オペランドがシンボル指定によるPC相対(symbol)である場合、アセンブラはPCの値を考慮した上でディスプレースメントを逆算し、オブジェクトコードを生成します。

11.3 DSP 命令

11.3.1 プログラムの要素

(1) ソースステートメント

SH2-DSP、SH3-DSP の命令は実行命令と DSP 命令に分類できます。DSP 命令は DSP レジスタを操作する命令です。DSP 命令は実行命令とは異なる命令体系を持ち、記述方法も異なります。

DSP 命令では 1 ステートメント内に複数のオペレーションを記述することができます。

DSP 命令のオペレーションには以下の種類があります。

(a) DSP演算オペレーション

DSPレジスタ間の演算を指定するオペレーションで、以下のものがあります。

PABS、PADD、PADDC、PAND、PCLR、PCMP、PCOPY、PDEC、PDMSB、PINC、PLDS、PMULS、PNEG、POR、PRND、PSHA、PSHL、PSTS、PSUB、PSUBC、PXOR

(b) Xデータ転送オペレーション

DSPレジスタとXデータメモリ間のデータ転送を指定するオペレーションで、以下のものがあります。

MOVX、NOPX

(c) Yデータ転送オペレーション

DSPレジスタとYデータメモリ間のデータ転送を指定するオペレーションで、以下のものがあります。

MOVY、NOPY

(d) シングルデータ転送オペレーション

Xデータメモリ、Yデータメモリに限定されない一般のメモリとDSPレジスタ間のデータ転送を指定するオペレーションで、以下のものがあります。

MOVS

(2) 並列演算命令

並列演算命令は、DSP 演算と同時に DSP レジスタと X データメモリ、Y データメモリ間のデータ転送を行なうことを指定します。命令のサイズは 32 ビットになります。

並列演算命令の構成を以下に示します。

```
<ラベル>[ <DSP 演算部>][ <データ転送部>][ <コメント>]
```

(a) DSP 演算部の記述方法

DSP 演算部の構成を以下に示します。

```
<コンディション> [<DSP 演算オペレーション> <オペランド>[ ...]]
```

• コンディション

コンディションは並列動作命令を実行する条件を指定します。

コンディションには次のものがあります。

11. アセンブラ言語仕様

DCT、DCF

DCT は DC ビットが 1 の時に命令を実行することを指定します。

DCF は DC ビットが 0 の時に命令を実行することを指定します。

• DSP 演算オペレーション

DSP 演算を指定します。DSP 演算オペレーションを 2 個指定できるのは PADD と PMULS、PSUB と PMULS の組み合わせを指定するときだけです。

(b) データ転送部の記述方法

データ転送部の構成を以下に示します。

```
<X データ転送オペレーション>[ <オペランド>]]  
[ <Y データ転送オペレーション>[ <オペランド>]]
```

X データメモリの転送と Y データメモリの転送をこの順に指定します。

データ転送のオペレーションが NOPX または NOPY の場合は省略することができます。

コーディング例

```
LABEL1: PADD A0,M0,A0 PMULS X0,Y0,M0 MOVX.W @R4+,X0 MOVS.W @R6+,Y0 ; DSP 命令  
ラベル DSP 演算部 データ転送部 コメント
```

```
DCT PINC X1,A1 MOVX.W @R4,X0 MOVS.W @R6+,Y0  
DSP 演算部 データ転送部
```

```
PCMP X1,M0 MOVX.W @R4,X0 ; Y メモリ転送を省略  
DSP 演算部 データ転送部 コメント
```

(3) データ転送命令

データ転送命令は X データメモリの転送と Y データメモリの転送の組み合わせ、またはシングルデータ転送を指定します。

(a) X データメモリの転送と Y データメモリの転送の組み合わせ

X データメモリの転送と Y データメモリの転送の組み合わせの構成を以下に示します。

```
<ラベル>[ <X データ転送オペレーション>[ <オペランド>]]  
[ <Y データ転送オペレーション>[ <オペランド>]] [<コメント>]
```

X データメモリの転送と Y データメモリの転送をこの順に指定します。データ転送のオペレーションが NOPX または NOPY の場合は省略することができます。ただし、並列演算命令の場合と異なり、X データメモリの転送と Y データメモリの転送の両方を省略することはできません。

X データメモリの転送命令と Y データメモリの転送命令の記述例を以下に示します。

コーディング例

```
LABEL2: MOVX.W @R4,X0 ; データ転送命令(Y データメモリ転送を省略)  
MOVX.W @R4,X0 MOVS.W @R6+,Y0
```

(b) シングルデータ転送命令

シングルデータ転送命令の構成を以下に示します。

```
[<ラベル>][ <シングルデータ転送オペレーション> <オペランド>][<コメント>]
```

MOVS 命令を指定します。

シングルデータ転送命令の記述例を以下に示します。

コーディング例

```
LABEL3: MOVS.W @-R2,A0 ; シングルデータ転送
```

(4) 複数行にわたるソースステートメントの書き方

DSP 命令は 1 行に複数のオペレーションを記述することができるため、ソースステートメントが長くなり、プログラムが見にくくなります。そこで、DSP 命令ではオペランドを区切るカンマ(,)以外にもオペランドとオペレーションの間で区切ることができます。

複数行にわたるソースステートメントは次の(a)~(c)の手順で記述してください。

- (a) オペランドとオペレーションを切れ目として改行します。
- (b) すぐ次の行の1カラム目にプラス(+を書きます。
- (c) そのプラスの後ろにソースステートメントの続きを書きます。

プラスの後ろに空白またはタブを入れてもかまいません。

コーディング例

```

                PADD                A0,M0,X0
+              PMULS               A1,Y1,M0
+              MOVX                 @R4,X0
+              MOVY                 @R6,Y1
```

; 1つのソースステートメントを4行にわたって書いた例です。

11.3.2 DSP 命令

(1) DSP 演算命令

表 11.18 に DSP 演算命令のニーモニックを示します。

表 11.18 DSP 演算命令ニーモニックの分類

分類	ニーモニック
DSP 算術演算命令	PADD, PSUB, PCOPY, PDMSB, PINC, PNEG, PMULS, PADDC, PSUBC, PCMP, PDEC, PABS, PRND, PCLR, PLDS, PSTS
DSP 論理演算命令	POR, PAND, PXOR
DSP シフト演算命令	PSHA, PSHL

(a) オペレーションサイズ

DSP 演算命令にオペレーションサイズは指定できません。

(b) アドレス形式

表 11.19 に DSP 演算命令のアドレス形式を示します。

表 11.19 DSP 演算命令のアドレス形式

アドレス形式	表記法
DSP レジスタ直接	Dp(DSP レジスタ名)
イミディエイトデータ	#imm

• DSP レジスタ直接

表 11.20 に DSP レジスタ直接に指定できるレジスタを示します。

表の中で Sx、Sy、Dz、Du、Se、Sf、Dg は「表 11.22 DSP 演算命令一覧」を参照してください。

表 11.20 DSP レジスタ直接に指定可能なレジスタ

	DSP レジスタ							
	A0	A1	M0	M1	X0	X1	Y0	Y1
Sx								
Sy								
Dz								
Dp Du								
Se								
Sf								
Dg								

- イミディエイトデータ

イミディエイトデータは PSHA、PSHL 命令の第 1 オペランドにのみ指定できます。指定できる値は以下の通りです。

- 値の種類
定数、シンボル、または式を指定することができます。
- シンボルの種類
イミディエイトデータには相対アドレスシンボルや外部参照シンボルを含めて、任意のシンボルが指定できます。
- 値の範囲
表 11.21 に指定できる値の範囲を示します。

表 11.21 イミディエイトデータの値の範囲

命令	数値の範囲
PSHA 命令	H'FFFFFFE0 ~ H'00000020(-32 ~ 32)
PSHL 命令	H'FFFFFFF0 ~ H'00000010(-16 ~ 16)

(c) 複数の DSP 演算命令の組み合わせ

PADD 命令と PMULS 命令、または PSUB 命令と PMULS 命令の組み合わせを指定することができます。この命令の組み合わせは本来一つの DSP 演算命令ですが、プログラムを読み易くするために PADD 命令または PSUB 命令のオペランドと PMULS 命令のオペランドを分割して記述できるようにしているものです。

複数の DSP 演算命令の組み合わせの例を以下に示します。

コーディング例

```
PADD A0,M0,A0 PMULS X0,Y0,M0 NOPX MOVY.W @R6+,Y0
PSUB A1,M1,A1 PMULS X1,Y1,M1 MOVX @R4+,X0 NOPY
```

【注】 複数の DSP 演算命令の組み合わせでディスティネーションレジスタとして同一のレジスタを指定した場合ウォーニング 701 になります。

```
PADD A0,M0,A0 PMULS X0,Y0,A0 ウォーニング 701
```

(d) コンディション付き DSP 演算命令

DSP 演算命令には DSR レジスタの DC ビットの状態により実行するかどうかを選択できるものがあります。コンディション DCT は DC ビットが 1 のときだけ命令を実行します。コンディション DCF は DC ビットが 0 のときだけ命令を実行します。

コンディションを付けられる DSP 演算命令には次のものがあります。

PADD、PAND、PCLR、PCOPY、PDEC、PDMSB、PINC、PLDS、PNEG、POR、PSHA、PSHL、PSTS、PSUB、PXOR

11. アセンブラ言語仕様

(e) DSP 演算命令一覧

表 11.22 に DSP 演算命令の一覧を示します。表中の Sx、Sy、Dz、Du、Se、Sf、Dg のところに指定できるレジスタは「表 11.20 DSP レジスタ直接に割り当て可能なレジスタ」を参照してください。

表 11.22 DSP 演算命令一覧

ニーモニック	アドレス形式	ニーモニック	アドレス形式
PABS	Sx,Dz	-	-
PABS	Sy,Dz	-	-
PADD	Sx,Sy,Dz	-	-
PADD	Sx,Sy,Du	PMULS	Se,Sf,Dg
PADDC	Sx,Sy,Dz	-	-
PAND	Sx,Sy,Dz	-	-
PCLR	Dz	-	-
PCMP	Sx,Sy	-	-
PCOPY	Sx,Dz	-	-
PCOPY	Sy,Dz	-	-
PDEC	Sx,Dz	-	-
PDEC	Sy,Dz	-	-
PDMSB	Sx,Dz	-	-
PDMSB	Sy,Dz	-	-
PINC	Sx,Dz	-	-
PINC	Sy,Dz	-	-
PLDS	Dz,MACH	-	-
PLDS	Dz,MACL	-	-
PMULS	Se,Sf,Dg	-	-
PNEG	Sx,Dz	-	-
PNEG	Sy,Dz	-	-
POR	Sx,Sy,Dz	-	-
PRND	Sx,Dz	-	-
PRND	Sy,Dz	-	-
PSHA	#imm,Dz	-	-
PSHA	Sx,Sy,Dz	-	-
PSHL	#imm,Dz	-	-
PSHL	Sx,Sy,Dz	-	-
PSTS	MACH,Dz	-	-
PSTS	MACL,Dz	-	-
PSUB	Sx,Sy,Dz	-	-
PSUB	Sx,Sy,Du	PMULS	Se,Sf,Dg
PSUBC	Sx,Sy,Dz	-	-
PXOR	Sx,Sy,Dz	-	-

(2) データ転送命令

(a) ニーモニック

データ転送命令にはデュアルメモリ転送とシングルメモリ転送があります。デュアルメモリ転送は X メモリと DSP レジスタ間、Y メモリと DSP レジスタ間のデータ転送を同時に行います。シングルメモリ転送では任意のメモリと DSP レジスタ間のデータ転送を行います。

表 11.23 にデータ転送命令のニーモニック一覧を示します。

表 11.23 データ転送命令のニーモニック一覧

分類	ニーモニック	
デュアルメモリ転送	X メモリ転送	NOPX
		MOVX
	Y メモリ転送	NOPY
		MOVY
シングルメモリ転送	MOVS	

(b) オペレーションサイズ

NOPX、NOPY 命令はサイズ指定できません。

MOVX、MOVY 命令はワードサイズ(W)の指定しかできません。サイズを指定しないとワードサイズと解釈します。

MOVS 命令はワードサイズ(W)とロングワードサイズ(L)の両方が指定できます。サイズを指定しないとロングワードサイズと解釈します。

(c) アドレス形式

表 11.24 にデータ転送命令で指定可能なアドレス形式を示します。

表 11.24 データ転送命令のアドレス形式

アドレス形式	表記法
DSP レジスタ直接	Dz
レジスタ間接	@Az
ポストインクリメントレジスタ間接	@Az+
インデックス付きレジスタ間接/ポストインクリメント	@Az+Iz
プリデクリメントレジスタ間接	@-Az

アドレス形式のうち、インデックス付きレジスタ間接/ポストインクリメントは DSP のデータ転送命令に特有なアドレス形式です。この形式はレジスタ Az の指す内容を参照した後、Az の内容を Iz の値だけインクリメントすることを示します。

(d) アドレス形式に指定可能なレジスタ

表 11.25 に DSP レジスタ直接、レジスタ間接、ポストインクリメントレジスタ間接、インデックス付きレジスタ間接/ポストインクリメント、プリデクリメント間接に指定可能なレジスタを示します。

表の中で Dx、Dy、Ds、Da、Ax、Ay、As、Ix、Iy、Is については「表 11.26 データ転送命令一覧」を参照してください。

11. アセンブラ言語仕様

表 11.25 データ転送命令のアドレス形式に指定可能なレジスタ

		汎用レジスタ								DSP レジスタ									
		R2	R3	R4	R5	R6	R7	R8	R9	A0	A1	M0	M1	X0	X1	Y0	Y1	A0G	A1G
Dz	Dx																		
	Dy																		
	Ds																		
	Da																		
Az	Ax																		
	Ay																		
	As																		
Iz	Ix																		
	Iy																		
	Is																		

【注】同一命令内の DSP 演算命令とデータ転送命令の間で同一のレジスタをディスティネーションに指定した場合はウォーニング 703 になります。

PADD A0,M0,Y0 NOPX MOVY.W @R6+,Y0 ウォーニング 703

(e) データ転送命令一覧

表 11.26 にデータ転送命令の一覧を示します。

表中の Dx、Dy、Ds、Da、Ax、Ay、As、Ix、Iy、Is のところに指定できるレジスタは「表 11.26 データ転送命令のアドレス形式に指定可能なレジスタ」を参照してください。

表 11.26 データ転送命令一覧

種別	ニーモニック	アドレス形式
X データ転送命令	NOPX	-
	MOVX.W	@Ax,Dx
	MOVX.W	@Ax+,Dx
	MOVX.W	@Ax+Ix,Dx
	MOVX.W	Da,@Ax
	MOVX.W	Da,@Ax+
	MOVX.W	Da,@Ax+Ix
Y データ転送命令	NOPLY	-
	MOVY.W	@Ay,Dy
	MOVY.W	@Ay+,Dy
	MOVY.W	@Ay+Iy,Dy
	MOVY.W	Da,@Ay
	MOVY.W	Da,@Ay+Iy
シングルデータ転送命令	MOV.S.W	@-As,Ds
	MOV.S.W	@As,Ds
	MOV.S.W	@As+,Ds
	MOV.S.W	@As+Is,Ds

種別	ニーモニック	アドレス形式
シングルデータ転送命令	MOVS.W	Ds, @-As
	MOVS.W	Ds, @As
	MOVS.W	Ds, @As+
	MOVS.W	Ds, @As+Is
	MOVS.L	@-As, Ds
	MOVS.L	@As, Ds
	MOVS.L	@As+, Ds
	MOVS.L	@As+Is, Ds
	MOVS.L	Ds, @-As
	MOVS.L	Ds, @As
	MOVS.L	Ds, @As+
	MOVS.L	Ds, @As+Is

11.4 アセンブラ制御命令

アセンブラ制御命令はアセンブラが解釈、実行する命令です。

書式の下線は、省略時の解釈を示します。

表 11.27 にアセンブラ制御命令の一覧を示します。

表 11.27 アセンブラ制御命令一覧

分類	二ーモニック	機能
CPUに関するもの	.CPU	CPU 種別を指定する
セクションまたは ロケーションカウンタ に関するもの	.SECTION	セクションを宣言する
	.ORG	ロケーションカウンタ値を設定する
	.ALIGN	ロケーションカウンタ値を境界調整数の倍数に補正する
シンボルに関するもの	.EQU	シンボルに値を設定する
	.ASSIGN	シンボルに値を設定または再設定する
	.REG	レジスタ別名を定義する
	.FREG	浮動小数点レジスタ名を定義する
データまたはデータ領域を 確保するもの	.DATA	整数データを確保する
	.DATAB	整数データブロックを確保する
	.SDATA	文字列データを確保する
	.SDATAB	文字列データブロックを確保する
	.SDATAC	計数付き文字列データを確保する
	.SDATAZ	ゼロ終端文字列データを確保する
	.FDATA	浮動小数点データを確保する
	.FDATAB	浮動小数点データブロックを確保する
	.XDATA	固定小数点データを確保する
	.RES	データ領域を確保する
	.SRES	文字列データ領域を確保する
	.SRESC	計数付き文字列データ領域を確保する
	.SRESZ	ゼロ終端文字列データ領域を確保する
	.FRES	浮動小数点データ領域を確保する
外部定義または外部参照に に関するもの	.EXPORT	外部定義シンボルを宣言する
	.IMPORT	外部参照シンボルを宣言する
	.GLOBAL	外部参照シンボルまたは外部定義シンボルを宣言する
オブジェクトモジュールに に関するもの	.OUTPUT	オブジェクトモジュール、デバッグ情報の出力を制御する
	.DEBUG	シンボルデバッグ情報の部分出力を制御する
	.ENDIAN	エンディアン種別を指定する
	.LINE	行番号を変更する
アセンブルリストに関する もの	.PRINT	アセンブルリストの出力を制御する
	.LIST	ソースプログラム・リストの部分出力を制御する
	.FORM	アセンブルリストの行数と桁数を設定する
	.HEADING	ソースプログラム・リストのヘッダを設定する
	.PAGE	ソースプログラム・リストを改ページする
	.SPACE	ソースプログラム・リストに空行を出力する
その他	.PROGRAM	オブジェクトモジュール名を設定する
	.RADIX	基数のない整数定数を何進数とするかを指定する
	.END	ソースプログラムの終わりとエントリポイントを指定する

【注】 コンパイラが出力するアセンブリソースには、.FILE が記述されます。.FILE はコンパイラ出力アセンブリソースにおいてのみ有効です。ユーザ作成アセンブリプログラムでは使用しないでください。

.CPU

書 式 .CPU <CPU 種別>

CPU 種別	対象 CPU
SH1	SH-1 用にあsembleする。
SH2	SH-2 用にあsembleする。
SH2E	SH-2E 用にあsembleする。
SH3	SH-3 用にあsembleする。
SH4	SH-4 用にあsembleする。
SHDSP	SH2-DSP 用にあsembleする。
SH3DSP	SH3-DSP 用にあsembleする。

ラベルは記述できません

- 説 明 アsembleするソースプログラムの対象とする CPU 種別を指定します。
 .CPU はソースプログラムの最初に記述してください。アsembleリストに関する制御命令を除いて、ソースプログラムの最初でない場合はエラーとなります。
 .CPU の指定は 1 回限り有効です。
 CPU 種別の優先順位は cpu オプション、.CPU、SHCPU 環境変数の順となります。
 指定を省略した場合、SHCPU 環境変数で設定した CPU 種別が有効となります。

例 .CPU SH2 ; SH-2 用にあsembleします。
 .SECTION A, CODE, ALIGN=4
 MOV.L R0, R1
 MOV.L R0, R2

.SECTION

書 式 .SECTION <セクション名>[,<セクション属性>[,<形式種別>]]
 <セクション属性> = { CODE |
 DATA |
 STACK |
 DUMMY }

 <形式種別> = { LOCATE = <先頭アドレス> |
 ALIGN = <境界調整数> }

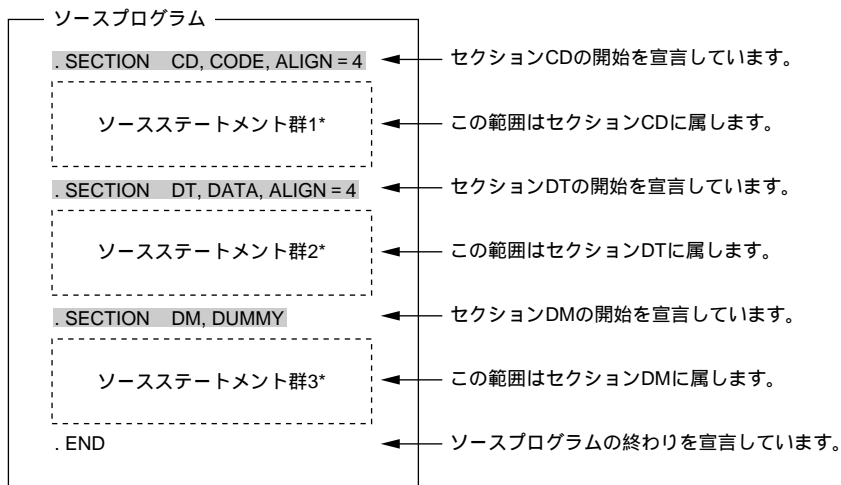
ラベルは記述できません。

説 明

セクションの宣言、再開を指定します。
 セクションとはプログラムの1区切りであり、リンケージの処理単位です。
 セクション属性は以下ようになります。

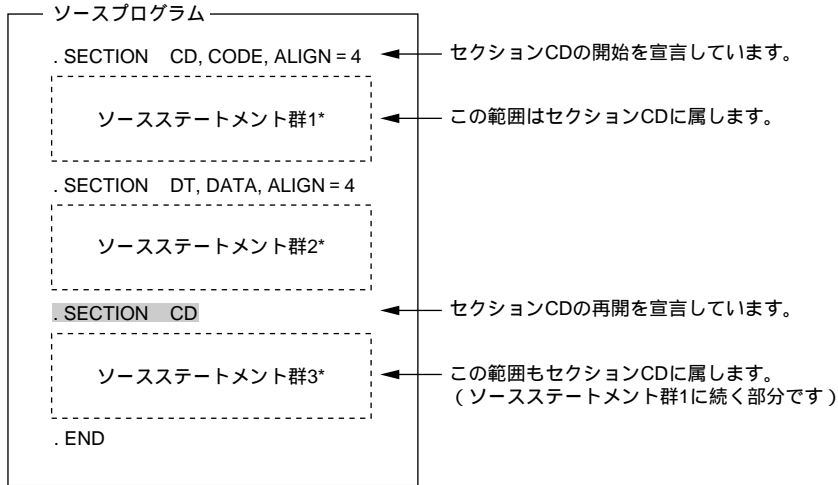
- CODE コードセクション
- DATA データセクション
- STACK スタックセクション
- DUMMY ダミーセクション

LOCATE=<先頭アドレス>を指定した場合、絶対アドレス形式でオブジェクトを出力します。
 ALIGN=<境界調整数>を指定した場合、相対アドレス形式でオブジェクトを出力します。
 最適化リンケージエディタは、そのセクションの先頭が境界調整数の倍数にあたる絶対アドレスにくるように調整します。
 形式種別の指定がない場合、ALIGN=4 が設定されます。
 セクションの宣言について簡単な例をあげて説明します。



【注】* この例では、「ソースステートメント群1~3」に .SECTIONが現れないことを仮定しています。

すでに宣言してあるセクションを同じファイルの中で再び宣言し、再開できます。
セクションの再開について、簡単な例をあげて説明します。



【注】* この例では、「ソースステートメント群1~3」に
.SECTIONが現れないことを仮定しています。

セクションを再開する場合、第 2 オペランドと第 3 オペランドの指定を省略してください(そのセクションを開始したときの指定がそのまま有効です)。

絶対アドレスセクションを開始するときには第 3 オペランドに「LOCATE=先頭アドレス」を指定してください。先頭アドレスはそのセクションが始まる絶対アドレスです。

先頭アドレスは次のように指定します。

- ・ 定数値を指定する。
かつ
- ・ 前方参照シンボルを使わずに指定する。

先頭アドレスとして許される値は H'00000000 ~ H'FFFFFFFF です。

(10 進表現では 0 ~ 4,294,967,295)

相対アドレスセクションを開始するときには第 3 オペランドに「ALIGN=境界調整数」を指定してください。リンケージエディタはそのセクションの先頭が境界調整数の倍数にあたる絶対アドレスにくるように調整します。

境界調整数は次のように指定します。

- ・ 定数値を指定する。
かつ
- ・ 前方参照シンボルを使わずに指定する。

境界調整数として許される値は 2 のべき乗 ($2^0, 2^1, 2^2, \dots, 2^{31}$) です。

コードセクションの場合は 4 以上の値 ($2^2, 2^3, 2^4, \dots, 2^{31}$) です。

次のいずれかの場合にアセンブラはデフォルトセクションを用意します。

- ・ セクションを宣言しないうちに実行命令、拡張命令、DSP 命令を記述している。
- ・ セクションを宣言しないうちにデータを確保するアセンブラ制御命令を記述している。
- ・ セクションを宣言しないうちに .ALIGN を記述している。
- ・ セクションを宣言しないうちに .ORG を記述している。
- ・ セクションを宣言しないうちにロケーションカウンタを参照している。
- ・ セクションを宣言しないうちにラベルだけの行を記述している。

11. アセンブラ言語仕様

デフォルトセクションとは次のようなセクションです。

- ・セクション名：P
- ・セクションの種類：コードセクション、相対アドレスセクション(境界調整数=4)

例

```

      .ALIGN      4          ; この範囲は、デフォルトセクションPに属します。
      .DATA.L     H' 11111111 ; デフォルトセクションPはコードセクションであり、
      ~           ; 境界調整数 = 4の相対アドレスセクションです。
;
; .SECTION CD, CODE, ALIGN = 4
;
      MOV         R0, R1     ; この範囲は、セクションCDに属します。
      MOV         R0, R2     ; セクションCDはコードセクションであり、
      ~           ; 境界調整数 = 4の相対アドレスセクションです。
;
; .SECTION DT, DATA, LOCATE = H' 00001000
;
X1 :   .DATA.L     H' 22222222 ; この範囲は、セクションDTに属します。
      .DATA.L     H' 33333333 ; セクションDTは、データセクションであり、
      ~           ; 先頭アドレス = H' 00001000の絶対アドレス
;                               ; セクションです。
      .END
```

この例では「～」の部分に .SECTION が現れないことを仮定しています。

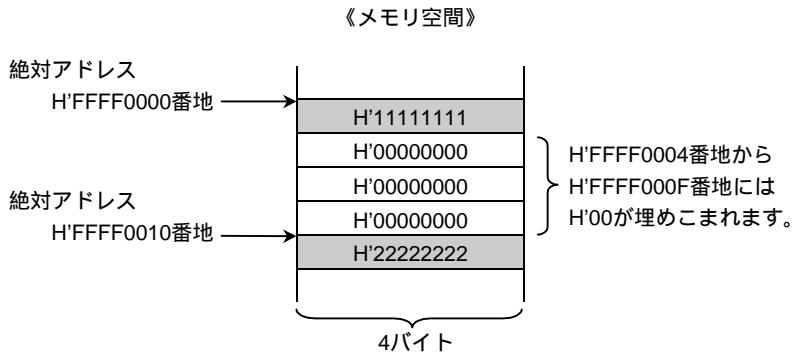
ロケーションカウンタ値設定

.ORG

書 式 .ORG <ロケーションカウンタ値>
 ラベルは記述できません。

説 明 セクション内のロケーションカウンタ値を指定した値に設定します。
 .ORG によって実行命令やデータを特定のアドレスに配置できます。
 ロケーションカウンタ値は次のように指定します。
 ・ 定数値またはセクション内のアドレスを指定する。
 かつ
 ・ 前方参照シンボルを使わずに指定する。
 ロケーションカウンタ値として許される値は H'00000000 ~ H'FFFFFFF です。
 (10 進表現では 0 ~ 4,294,967,295)
 絶対アドレスセクションで指定する場合は、
 ロケーションカウンタ値 セクション先頭アドレス (符号なしの値として比較)
 となるようにしてください。
 アセンブラはロケーションカウンタ値を次のように見なします。
 ・ 絶対アドレスセクション内では絶対アドレス
 ・ 相対アドレスセクション内では相対アドレス (セクション先頭からの相対的な距離)

例 .SECTION DT,DATA,LOCATE=H'FFFF0000
 .DATA.L H'11111111
 .ORG H'FFFF0010 ;ロケーションカウンタ値を設定しています。
 .DATA.L H'22222222 ;整数データ H'22222222 を絶対アドレスの
 ~ ;H'FFFF0010 番地に確保しています。



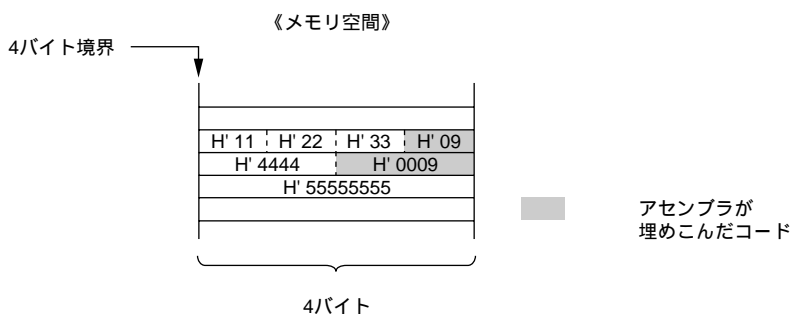
.ALIGN

書 式 .ALIGN <境界調整数>
 ラベルは記述できません。

説 明 セクション内のロケーションカウンタ値を境界調整数の倍数に補正します。
 .ALIGN によって実行命令やデータを特定の境界(アドレスの区切り)に配置できます。
 ロケーションカウンタ値は次のように指定します。
 ・ 定数値を指定する。
 かつ
 ・ 前方参照シンボルを使わずに指定する。
 境界調整数として許される値は 2 のべき乗 ($2^0, 2^1, 2^2, \dots, 2^{31}$) です。
 相対アドレスセクションで .ALIGN を使用する場合は
 . SECTION で指定する境界調整数 .ALIGN で指定する境界調整数
 となるようにしてください。
 コードセクション、データセクションに .ALIGN を記述すると、アセンブラは NOP 命令のオブジェクトコードをメモリ上に埋めこみ、ロケーションカウンタ値を補正します。半端なバイトサイズの領域には H'09 を埋めこみます。
 【注】* このようなオブジェクトコードはアSEMBルリスト上に表れません。

例 . SECTION P, CODE, ALIGN=4
 .DATA.B H'11
 .DATA.B H'22
 .DATA.B H'33
 .ALIGN 2 ;ロケーションカウンタ値を 2 の倍数に補正しています。
 .DATA.W H'4444
 .ALIGN 4 ;ロケーションカウンタ値を 4 の倍数に補正しています。
 .DATA.L H'55555555
 ~

バイトサイズの整数データ H'11 がもともと 4 バイト境界に位置するものと仮定します。
アセンブラは下図のようにオブジェクトコードを埋めこんで境界調整します。



シンボル値設定

.EQU

書 式 <シンボル>[:] .EQU <シンボル値>

説 明 シンボルに値を設定します。
 .EQU で定義したシンボルは再定義できません。
 シンボル値は次のように指定します。

- ・ 定数値、アドレス、外部参照シンボルの値を指定する。
 かつ
- ・ 前方参照シンボルを使わずに指定する。

シンボル値として許される値は H'00000000 ~ H'FFFFFFF です。
 (10 進表現では -2,147,483,648 ~ 4,294,967,295)

【注】* 外部参照シンボル、外部参照シンボル+定数、外部参照シンボル-定数が記述できます。

例

```

~
X1: .EQU      10           ;X1 の値は 10 になります。
X2: .EQU      20           ;X2 の値は 20 になります。
    CMP/EQ    #X1,R0      ;CMP/EQ #10,R0 と同じです。
    BT        LABEL1
    CMP/EQ    #X2,R0      ;CMP/EQ #20,R0 と同じです。
    BT        LABEL2
~

```

.ASSIGN

書 式 <シンボル>[:] .ASSIGN <シンボル値>

- 説 明 シンボルに値を設定します。
 .ASSIGN で定義したシンボルは .ASSIGN で再定義できます。
 シンボル値は次のように指定します。
- ・ 定数値またはアドレスを指定する。
 かつ
 - ・ 前方参照シンボルを使わずに指定する。
- シンボル値として許される値は H'00000000 ~ H'FFFFFFF です。
 (10 進表現では -2,147,483,648 ~ 4,294,967,295)
 .ASSIGN による定義は定義した位置から有効です。
 .ASSIGN で定義したシンボルには次の使用上の制限があります。
- ・ 外部参照または外部定義できない。
 - ・ デバッガで参照できない。

例

```

~
X1: .ASSIGN 1
X2: .ASSIGN 2
    CMP/EQ #X1,R0 ;CMP/EQ #1,R0 と同じです。
    BT LABEL1
    CMP/EQ #X2,R0 ;CMP/EQ #2,R0 と同じです。
    BT LABEL2
~
X1: .ASSIGN 3
X2: .ASSIGN 4
    CMP/EQ #X1,R0 ;CMP/EQ #3,R0 と同じです。
    BT LABEL3
    CMP/EQ #X2,R0 ;CMP/EQ #4,R0 と同じです。
    BF LABEL4
~

```

レジスタ別名定義

.REG

書 式 <シンボル>[:] .REG <レジスタ名>
 または
 <シンボル>[:] .REG (<レジスタ名>)

説 明 レジスタ別名を定義します。
 .REG で定義したレジスタ別名は元のレジスタ名と全く同等に使用できます。
 .REG で定義したレジスタ別名は再定義できません。
 別名をつけることができるのは汎用レジスタ (R0 ~ R15, SP) だけです。
 .REG による定義は定義した位置から有効です。
 .REG で定義したシンボルには次の使用上の制限があります。
 ・ 外部参照または外部定義できない。

例

```

~
MIN:  .REG      R10
MAX:  .REG      R11
      MOV      #0,MIN          ;MOV #0,R10 と同じです。
      MOV      #99,MAX        ;MOV #99,R11 と同じです。
      CMP/HS   MIN,R1
      BF      LABEL
      CMP/HS   R1,MAX
      BF      LABEL
~

```

.FREG

書 式	<シンボル>[:] .FREG <浮動小数点レジスタ名> または <シンボル>[:] .FREG (<浮動小数点レジスタ名>)
説 明	<p>浮動小数点レジスタ別名を定義します。</p> <p>.FREG で定義した浮動小数点レジスタ名は元のレジスタ名と全く同等に使用できます。</p> <p>.FREG で定義したレジスタ別名は再定義できません。</p> <p>別名をつけることができるのは以下の浮動小数点レジスタです。</p> <ul style="list-style-type: none"> • FRm (m = 0 ~ 15) • DRn (n = 0, 2, 4, 6, 8, 10, 12, 14) • XDn (n = 0, 2, 4, 6, 8, 10, 12, 14) • Fvi (i = 0, 4, 8, 12) <p>.FREG による定義は定義した位置から有効です。</p> <p>.FREG で定義したシンボルには次の使用上の制限があります。</p> <ul style="list-style-type: none"> • 外部参照または外部定義できない。 <p>.FREG は CPU 種別が SH2E、SH4 のとき使用できます。</p>
例	<pre> ~ MAX: .FREG FR11 FMOV FR1,MAX ;FMOV FR1,FR11 と同じです。 FCMP/EQ MAX,FR2 ;FCMP/EQ FR11,FR2 と同じです。 BF LABEL ~ </pre>

整数データ確保

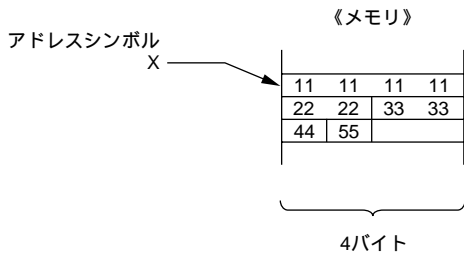
.DATA

書式 [<シンボル>[:]] .DATA[.オペレーションサイズ] 整数データ[,...]
 <オペレーションサイズ> = { B | W | L }

説明 整数データをメモリ上に確保します。
 オペレーションサイズによって確保するデータのサイズが決まります。
 オペレーションサイズを省略するとロングワードになります。
 整数データには相対アドレス、外部参照シンボル、前方参照シンボルを含めて任意の値を指定できます。
 オペレーションサイズおよび整数データの範囲は、次のようになります。

サイズ	データのサイズ	整数データの範囲(10進表現)
B	バイト (1バイト)	H'00000000~H'000000FF (0~255)
		H'FFFFFF80~H'FFFFFFF (-128~-1)
W	ワード (2バイト)	H'00000000~H'0000FFFF (0~65,535)
		H'FFFF8000~H'FFFFFFF (-32,768~-1)
L	ロングワード (4バイト)	H'00000000~H'FFFFFFF (0~4,294,967,295)
		H'80000000~H'FFFFFFF (-2,147,483,648~-1)

例 ~
 .ALIGN 4
 X:
 .DATA.L H'11111111 ;
 .DATA.W H'2222,H'3333 ;整数データを確保しています。
 .DATA.B H'44,H'55 ;
 ~



【注】データは16進数です。

文字列データ確保

.SDATA

書 式 [<シンボル>[:]] .SDATA "<文字列>"[,...]

説 明 文字列データをメモリ上に確保します。
 文字列に制御文字をつけ加えることができます。
 記述形式は次のとおりです。
 "文字列"<制御文字の ASCII コード>
 制御文字の ASCII コードは次のように指定します。

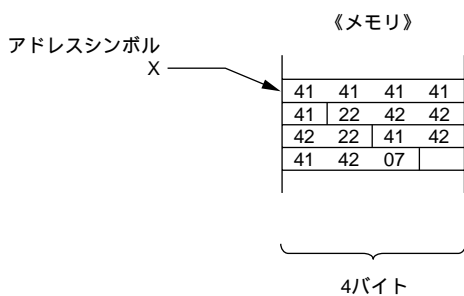
- ・ 定数値を指定する。
 かつ
- ・ 前方参照シンボルを使わずに指定する。

例

```

~
.ALIGN 4
X:
.SDATA "AAAA" ;文字列データを確保しています。
.SDATA "" "BBB" "" ;ダブルコーテーションを含む例です。
.SDATA "ABAB" <H'07> ;制御文字をつけ加えた例です。
~

```



【注】1 データは16進数です。

【注】2 文字AのASCIIコード H' 41
 文字BのASCIIコード H' 42
 文字 " のASCIIコード H' 22

.SDATAB

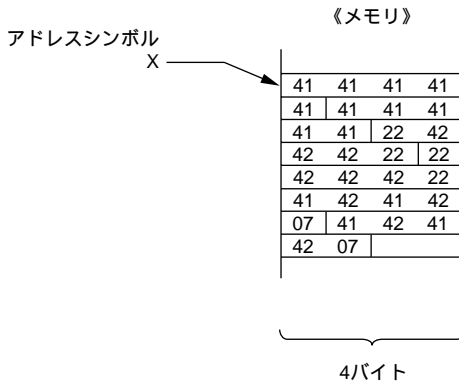
書 式 [<シンボル>[:]] .SDATAB <ブロック数>,"<文字列>"

説 明 文字列データを指定の数だけ連続してメモリ上に確保します。
 ブロック数は次のように指定します。
 ・定数値を指定する。
 かつ
 ・前方参照シンボルを使わずに指定する。
 ブロック数には 1 以上の値を指定してください。
 ブロック数の上限値は文字列データの長さによって決まります。
 (文字列データの長さ×ブロック数 H'FFFFFFFF (4,294,967,295) バイト)
 文字列に制御文字をつけ加えることができます。
 記述形式は次のとおりです。
 "文字列"<制御文字の ASCII コード>
 制御文字の ASCII コードは次のように指定します。
 ・定数値を指定する。
 かつ
 ・前方参照シンボルを使わずに指定する。

例

```

~
.ALIGN 4
X:
.SDATAB 2,"AAAA" ;文字列データブロックを確保しています。
.SDATAB 2," "BBB" " ;ダブルコーテーションを含む例です。
.SDATAB 2,"ABAB"<H'07> ;制御文字をつけ加えた例です。
~
    
```



【注】1 データは16進数です。

【注】2 文字AのASCIIコード H' 41
 文字BのASCIIコード H' 42
 文字 " のASCIIコード H' 22

計数付き文字列データ確保

.SDATAC

書 式 [<シンボル>[:]] .SDATAC "<文字列>" [, ...]

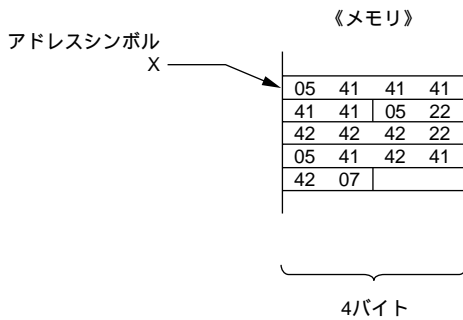
説 明 計数付き文字列データをメモリ上に確保します。
 計数付き文字列とは文字列の先頭に 1 バイトの計数をつけ加えたものです。
 計数は文字列データ(計数を含まないデータ)のサイズをバイト単位で表します。
 文字列に制御文字をつけ加えることができます。
 記述形式は次のとおりです。
 "文字列"<制御文字の ASCII コード>
 制御文字の ASCII コードは次のように指定します。
 ・ 定数値を指定する。
 かつ
 ・ 前方参照シンボルを使わずに指定する。

例

```

~
.ALIGN 4
X:
.SDATAC "AAAA" ;計数付き文字列データを確保しています。
.SDATAC " "BBB" " ;ダブルコーテーションを含む例です。
.SDATAC "ABAB" <H'07> ;制御文字をつけ加えた例です。
~

```



【注】1 データは16進数です。

【注】2 文字AのASCIIコード H' 41
 文字BのASCIIコード H' 42
 文字 " のASCIIコード H' 22

.SDATAZ

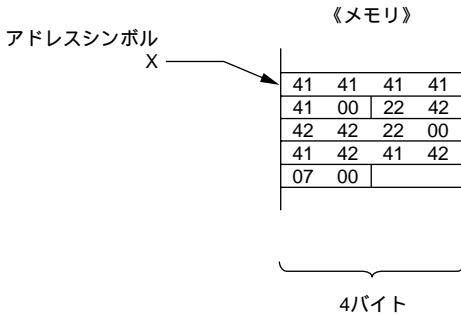
書 式 [<シンボル>[:]] .SDATAZ "<文字列>"[,...]

説 明 ゼロ終端文字列データをメモリ上に確保します。
 ゼロ終端文字列とは文字列の最後に 1 バイトのゼロをつけ加えたものです。
 文字列に制御文字をつけ加えることができます。
 記述形式は次のとおりです。
 "文字列"<制御文字の ASCII コード>
 制御文字の ASCII コードは次のように指定します。
 ・ 定数値を指定する。
 かつ
 ・ 前方参照シンボルを使わずに指定する。

例

```

~
.ALIGN 4
X:
.SDATAZ "AAAA" ;ゼロ終端文字列データを確保しています。
.SDATAZ " "BBB" " ;ダブルコーテーションを含む例です。
.SDATAZ "ABAB"<H'07> ;制御文字をつけ加えた例です。
~
    
```



【注】1 データは16進数です。

【注】2 文字AのASCIIコード H' 41
 文字BのASCIIコード H' 42
 文字 " のASCIIコード H' 22

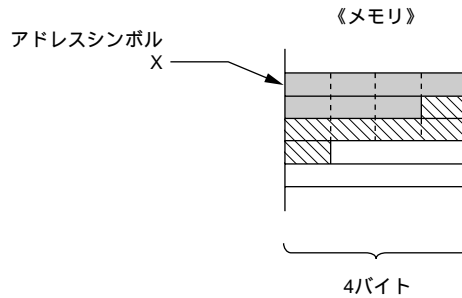
文字列データ領域確保

.SRES

書 式 [<シンボル>[:]] .SRES <文字列領域サイズ>[,...]

説 明 文字列用のデータ領域を確保します。
 文字列領域サイズは次のように指定します。
 ・ 定数値を指定する。
 かつ
 ・ 前方参照シンボルを使わずに指定する。
 文字列領域サイズとして許される値は H'00000001 ~ H'FFFFFFFF です。
 (10 進表現では 1 ~ 4,294,967,295)

例 ~
 .ALIGN 4
 X:
 .SRES 7 ;7 バイトの領域を確保しています。
 .SRES 6 ;6 バイトの領域を確保しています。
 ~



.SRESC

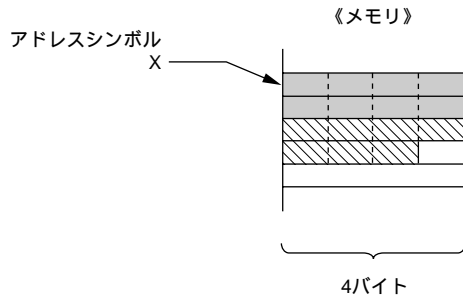
書 式 [<シンボル>[:]] .SRESC <文字列領域サイズ>[,...]

説 明 計数付き文字列用のデータ領域をメモリ上に確保します。
 計数付き文字列とは文字列の先頭に 1 バイトの計数をつけ加えたものです。
 計数は文字列用のデータ領域 (計数を含まない領域) のサイズをバイト単位で表します。
 文字列領域サイズは次のように指定します。
 ・ 定数値を指定する。
 かつ
 ・ 前方参照シンボルを使わずに指定する。
 文字列領域サイズとして許される値は H'00000000 ~ H'000000FF です。
 (10 進表現では 0 ~ 255)
 メモリ上に確保される領域のサイズは文字列領域サイズ + 計数用の 1 バイトです。

例

```

~
.ALIGN 4
X:
.SRESC 7 ;7 バイト + 計数用の 1 バイトを確保しています。
.SRESC 6 ;6 バイト + 計数用の 1 バイトを確保しています。
~
    
```



ゼロ終端文字列データ領域確保

.SRESZ

書 式 [<シンボル>[:]] .SRESZ <文字列領域サイズ>[,...]

説 明 ゼロ終端文字列用のデータ領域をメモリ上に確保します。
 ゼロ終端文字列とは文字列の終端に 1 バイトのゼロをつけ加えたものです。
 文字列領域サイズは次のように指定します。

- ・ 定数値を指定する。
- かつ
- ・ 前方参照シンボルを使わずに指定する。

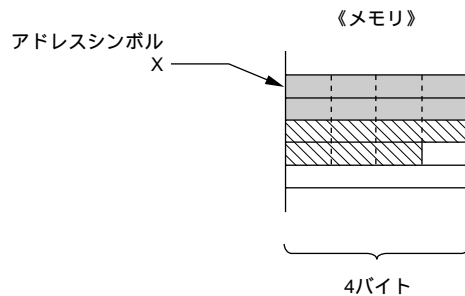
文字列領域サイズとして許される値は H'00000000 ~ H'000000FF です。
 (10 進表現では 0 ~ 255)
 メモリ上に確保される領域のサイズは文字列領域サイズ + 終端ゼロ用の 1 バイトです。

例

```

~
.SALIGN 4
X:
.SRESZ 7 ;7 バイト + 終端ゼロ用の 1 バイトを確保しています。
.SRESZ 6 ;6 バイト + 終端ゼロ用の 1 バイトを確保しています。
~

```



外部定義シンボル宣言

.EXPORT

書 式	<code>.EXPORT <シンボル>[,...]</code> ラベルは記述できません。
説 明	<p>外部定義シンボルを宣言します。 外部定義シンボルの宣言はファイル内で定義しているシンボルを他のファイルで参照するために必要です。 外部定義シンボルとして宣言できるのは次のものです。</p> <ul style="list-style-type: none"> ・定数シンボル(.ASSIGN で定義したものを除く) ・絶対アドレスシンボル(ダミーセクションのアドレスシンボルを除く) ・相対アドレスシンボル <p>シンボルを外部参照するには外部定義シンボルとして宣言するとともに外部参照シンボルとして宣言しなければなりません。 外部参照シンボルはシンボルを参照しているファイルで .IMPORT または .GLOBAL によって宣言します。</p>
例	<p>ファイル A で定義しているシンボルをファイル B で参照する例です。</p> <p>・ファイル A</p> <pre> .EXPORT X ;X を外部定義シンボルとして宣言します。 ~ X: .EQU H'10000000 ;X を定義します。 ~ </pre> <p>・ファイル B</p> <pre> .IMPORT X ;X を外部参照シンボルとして宣言します。 ~ .ALIGN 4 .DATA.L X ;X を参照します。 ~ </pre>

.IMPORT

書 式 .**IMPORT** <シンボル>[,...]
ラベルは記述できません。

説 明 外部参照シンボルを宣言します。
外部参照シンボルの宣言は他のファイルで定義しているシンボルを参照するために必要です。
ファイル内で定義しているシンボルは外部参照シンボルとして宣言できません。
シンボルを外部参照するには外部参照シンボルとして宣言するとともに外部定義シンボルとして宣言しなければなりません。
外部定義シンボルはシンボルを定義しているファイルで.**EXPORT**または.**GLOBAL**によって宣言します。

例 ファイル A で定義しているシンボルをファイル B で参照する例です。

```

・ファイル A
    .EXPORT      X                ;X を外部定義シンボルとして宣言します。
    ~
X:    .EQU      H'10000000        ;X を定義します。
    ~

・ファイル B
    .IMPORT      X                ;X を外部参照シンボルとして宣言します。
    ~
    .ALIGN      4
    .DATA.L     X                ;X を参照します。
    ~

```

外部定義シンボル、外部参照シンボル宣言

.GLOBAL

書 式	<code>.GLOBAL <シンボル>[,...]</code> ラベルは記述できません。
説 明	<p>外部定義シンボルまたは外部参照シンボルを宣言します。 外部定義シンボルの宣言はファイル内で定義しているシンボルを他のファイルで参照するために必要です。外部参照シンボルの宣言は他のファイルで定義しているシンボルをファイル内で参照するために必要です。 ファイル内で定義しているシンボルを <code>.GLOBAL</code> で宣言するとそのシンボルは外部定義シンボルになります。 ファイル内で定義していないシンボルを <code>.GLOBAL</code> で宣言するとそのシンボルは外部参照シンボルになります。 外部定義シンボルとして宣言できるのは、次のものです。</p> <ul style="list-style-type: none"> ・ 定数シンボル (<code>.ASSIGN</code> で定義したものを除く) ・ 絶対アドレスシンボル (ダミーセクションのアドレスシンボルを除く) ・ 相対アドレスシンボル <p>シンボルを外部参照するには外部定義シンボルとして宣言するとともに外部参照シンボルとして宣言しなければなりません。 外部定義シンボルはシンボルを定義しているファイルで <code>.EXPORT</code> または <code>.GLOBAL</code> によって宣言します。 外部参照シンボルはシンボルを参照しているファイルで <code>.IMPORT</code> または <code>.GLOBAL</code> によって宣言します。</p>
例	<p>ファイル A で定義しているシンボルをファイル B で参照する例です。</p> <p>・ ファイル A</p> <pre> .GLOBAL X ;X を外部定義シンボルとして宣言します。 ~ X: .EQU H'10000000 ;X を定義します。 ~ </pre> <p>・ ファイル B</p> <pre> .GLOBAL X ;X を外部参照シンボルとして宣言します。 ~ .ALIGN 4 .DATA.L X ;X を参照します。 ~ </pre>

オブジェクトモジュール、デバッグ情報出力制御

.OUTPUT

書 式 .OUTPUT <出力指定>[,...]
 <出力指定> = { OBJ | NOOBJ |
 DBG | NODBG }
 ラベルは記述できません。

説 明 オブジェクトモジュールまたはデバッグ情報の出力を制御します。

- (1) オブジェクトモジュールの出力
 オブジェクトモジュールの出力を制御します。
 出力種別は次のようになります。

出力種別	出力制御
OBJ	出力
NOOBJ	出力抑止

- (2) デバッグ情報の出力
 デバッグ情報の出力を制御します。
 出力種別は次のようになります。

出力種別	出力制御
DBG	出力
NODBG	出力抑止

本指定は、オブジェクトモジュールを出力時に有効です。

.OUTPUT を 2 回以上使用して指定内容が矛盾するとエラーとなります。

【例】

```
~
.OUTPUT OBJ
.OUTPUT NODBG
~
```

OK

```
~
.OUTPUT OBJ
.OUTPUT NOOBJ
~
```

エラー

デバッグ情報の出力に関する指定はオブジェクトモジュールを出力する場合に限り有効です。オブジェクトモジュールとデバッグ情報の出力に関して、アセンブラはオプションによる指定を優先します。

例 オブジェクトモジュールとデバッグ情報の出力に関して、オプションによる指定がないことを仮定して結果を説明しています。

- ・例 1

```
.OUTPUT    OBJ                   ;オブジェクトモジュールを出力します。
            ~                   ;デバッグ情報は出力しません。
~
```
- ・例 2

```
.OUTPUT    OBJ,DBG               ;オブジェクトモジュールとデバッグ情報を
            ~                   ;出力します。
~
```


・例 3

```
.OUTPUT      OBJ,NODBG      ;オブジェクトモジュールを出力します。  
              ;デバッグ情報は出力しません。  
~
```

備 考

デバッグ情報はデバッガでプログラムをデバッグするときに必要な情報であり、オブジェクトモジュールの一部となります。
デバッグ情報はソースステートメントの行に関する情報、シンボルに関する情報などを含みます。

.DEBUG

書 式 .DEBUG <出力指定>
 <出力指定> = { ON | OFF }
 ラベルは記述できません。

説 明 シンボルデバッグ情報の部分出力を制御します。
 デバッグに必要なシンボルに限定してシンボルデバッグ情報を出力するとアセンブル時間を短縮できるなどの利点があります。
 .DEBUG による指定はオブジェクトモジュールを出力し、かつ、デバッグ情報を出力する場合に限り有効です。
 出力種別は、以下のようになります。

出力種別	出力制御
ON	出力
OFF	出力抑止

例 ~
 .DEBUG OFF ;アセンブラは次のソースステートメント
 ;からシンボルデバッグ情報を出力しません。
 ~
 .DEBUG ON ;アセンブラは次のソースステートメント
 ;からシンボルデバッグ情報を出力します。
 ~

備 考 シンボルデバッグ情報とはデバッグ情報のうちのシンボルに関するものをいいます。

メモリのバイト並び順

.ENDIAN

書 式 .ENDIAN <エンディアン種別>
 <エンディアン種別> = { BIG | LITTLE }
 ラベルは記述できません。

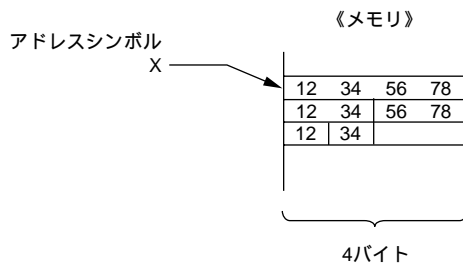
説 明 ターゲットマイコンのバイト並び順が Big Endian か Little Endian かを指定します。
 .ENDIAN はソースプログラムの最初に記述してください。
 エンディアン種別は、オプションによる指定を優先します。

指定内容	出力の制御
<u>BIG</u>	Big Endian でアセンブルする。
LITTLE	Little Endian でアセンブルする。

例 ・例 1 (Big Endian を指定した場合)

```
.CPU          SH3
.ENDIAN       BIG           ;Big Endian を指定します。
~

X:
.DATA.L      H'12345678    ;
.DATA.W      H'1234,H'5678 ;整数データを確保しています
.DATA.B      H'12,H'34     ;
~
```



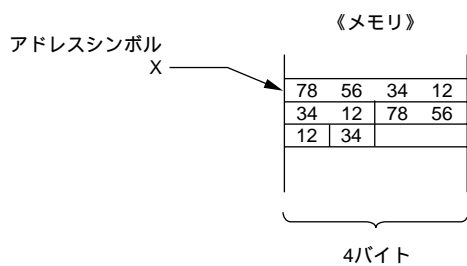
【注】 データは16進数です。

11. アセンブラ言語仕様

・例 2(Little Endian を指定した場合)

```
.CPU          SH3
.ENDIAN       LITTLE           ;Little Endian を指定します。
~

X:
.DATA.L      H'12345678       ;
.DATA.W      H'1234,H'5678    ;整数データを確保しています
.DATA.B      H'12,H'34        ;
~
```



【注】 データは16進数です。

.LINE

書 式 .LINE ["<ファイル名>",]<行番号>
ラベルは記述できません。

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 (v3.0 以降) は、デバッグオプションを指定してアセンブラソースを出力する時に C 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 shc -code=asmcode -debug test.c

Cソースプログラム(test.c)

```
int func()
{
    int i, j;

    j=0;
    for (i=1; i<=10; i++){
        j+=i;
    }
    return(j);
}
```

アセンブリソースプログラム(test.src)

```
.FILE      "C:\Yasm\test.c"
.EXPORT    _func
.SECTION   P, CODE, ALIGN=4

_func:
           ; function: func
           ; frame size=0
.LINE     "C:\Yasm\test.c", 5
MOV       #0, R5      ; H'00000000
.LINE     "C:\Yasm\test.c", 6
MOV       #1, R2      ; H'00000001
MOV       #10, R6     ; H'0000000A

L11:
ADD       #-1, R6
TST       R6, R6
.LINE     "C:\Yasm\test.c", 7
ADD       R2, R5
ADD       #1, R2
BF        L11
RTS
MOV       R5, R0
.END
```


ソースプログラムリスト部分出力制御

.LIST

書 式 .LIST <出力指定> [,...]
 <出力指定> = { ON | OFF |
 COND | NOCOND | DEF | NODEF | CALL | NOCALL |
 EXP | NOEXP | CODE | NOCODE }

ラベルは記述できません。

説 明

出力指定により次のような働きをします。

- (1) ソースステートメントの部分的な出力、出力抑止の制御
 - (2) 条件つきアセンブリ機能およびマクロ機能に関するソースステートメントの部分的な出力、出力抑止の制御
 - (3) オブジェクトコード表示行の部分的な出力、出力抑止の制御
- 各出力指定により制御される内容は以下のとおりです。

項目	出力指定		意味	制御内容
	出力	出力抑止		
(1)	<u>ON</u>	OFF	ソースステートメントの出力制御	本命令以降のソースステートメント
(2)	<u>COND</u>	NOCOND	条件つき不成立の出力制御 ^{*1}	.AIF, .AIFDEF の不成立部分
	<u>DEF</u>	NODEF	定義の出力制御 ^{*1}	マクロ定義部分 .AREPEAT, .AWHILE 定義部分 .INCLUDE .ASSIGNA, .ASSIGNC
	<u>CALL</u>	NOCALL	コールの出力制御 ^{*1}	マクロコール文 .AIF, .AIFDEF, .AENDI
(3)	<u>EXP</u>	NOEXP	展開の出力制御 ^{*1}	マクロ展開部分 .AREPEAT, .AWHILE 展開部分
	<u>CODE</u>	NOCODE	オブジェクトコード表示行の出力制御 ^{*1}	制御命令のオブジェクトコード表示が、ソースステートメントの行数を超える部分

【注】*1 本指定は、show/noshow オプションの指定がない場合に有効です。

.LIST による指定はソースプログラムリストを出力する場合に限り有効です。

ソースプログラム・リストの部分出力に関して、アセンブラはオプションによる指定を優先します。

.LIST 自体はソースプログラムリスト上に表示されません。

例

ソースプログラム・リストの部分出力に関して、コマンドライン・オプションによる指定がないことを仮定して結果を説明しています。

11. アセンブラ言語仕様

	.LIST	NOCOND,NODEF		
	.MACRO	SHLRN COUNT,Rd		ソースプログラムリストの 部分出力を制御します。
			:	
SHIFT	.ASSIGNA	¥COUNT	:	
			:	
	.AIF	¥&SHIFT GE 16	:	
	SHLR16	¥Rd	:	
SHIFT	.ASSIGNA	¥&SHIFT-16	:	
	.AENDI		:	
			:	
	.AIF	¥&SHIFT GE 8	:	
	SHLR8	¥Rd	:	
SHIFT	.ASSIGNA	¥&SHIFT-8	:	
	.AENDI		:	
			:	
	.AIF	¥&SHIFT GE 4	:	汎用多ビットシフトを
	SHLR2	¥Rd	:	マクロ定義しています。
	SHLR2	¥Rd	:	
SHIFT	.ASSIGNA	¥&SHIFT-4	:	
	.AENDI		:	
			:	
	.AIF	¥&SHIFT GE 2	:	
	SHLR2	¥Rd	:	
SHIFT	.ASSIGNA	¥&SHIFT-2	:	
	.AENDI		:	
			:	
	.AIF ¥&SHIFT GE 1		:	
	SHLR	¥Rd	:	
	.AENDI		:	
	.ENDM		:	
			:	
	SHLRN	23,R0		マクロコール

コーディング例のソースリスト出力結果

.LIST によってマクロ定義、.ASSIGNA、.ASSIGNC、.AIF または .AIFDEF の不成立部分の出力が抑止されています。

31	31
32	32 SHLRN 23,R0
33	M
35	M
36	M .AIF 23 GE 16
37 00000000 4029	C SHLR16 R0
39	M .AENDI
40	M
41	M .AIF 7 GE 8
45	M
46	M .AIF 7 GE 4
47 00000002 4009	C SHLR2 R0
48 00000004 4009	C SHLR2 R0
50	M .AENDI
51	M
52	M .AIF 3 GE 2
53 00000006 4009	C SHLR2 R0
55	M .AENDI
56	M
57	M .AIF 1 GE 1
58 00000008 4001	C SHLR R0
59	M .AENDI

.FORM

書 式 .FORM <サイズ指定>[,...]
 <サイズ指定> = { LIN = <行数> | COL = <桁数> }
 ラベルは記述できません。

説 明 アセンブリリストの 1 ページあたりの行数と 1 行あたりの桁数を設定します。
 行数と桁数は次のように指定します。
 ・ 定数値を指定する。
 かつ
 ・ 前方参照シンボルを使わずに指定する。
 <行数>、<桁数>の許される値の範囲、および .FORM による指定もオプションによる指定もない場合の解釈は次の通りです。

指定内容	意味	許される値 ^{*3}	未指定時
LIN=<行数>	1 ページあたりの行数 ^{*1}	20 ~ 255	60
COL=<桁数>	1 行あたりの桁数 ^{*2}	79 ~ 255	132

【注】 *1 lines オプションの指定がない場合、有効になります。

*2 columns オプションの指定がない場合、有効になります。

*3 範囲外の値を指定した時は、20 より小さい場合は 20、255 より大きい場合は 255 が指定されます。この場合、エラーは表示されません。

アセンブリリストの行数と桁数に関して、アセンブラはオプションによる指定を優先します。
 .FORM は 1 つのソースプログラムで何回でも使えます。

例 アセンブリリストの行数と桁数の設定に関して、オプションによる指定がないことを仮定して結果を説明しています。

```

~
  .FORM LIN=60, COL=200           ;このページからアセンブリリスト 1 ページ
                                   ;を 60 行にします。
                                   ;また、この行からアセンブリリスト 1 行を
                                   ;200 桁にします。
~
  .FORM LIN=55, COL=150          ;このページからアセンブリリスト 1 ページ
                                   ;を 55 行にします。
                                   ;また、この行からアセンブリリスト 1 行を
                                   ;150 桁にします。
~

```

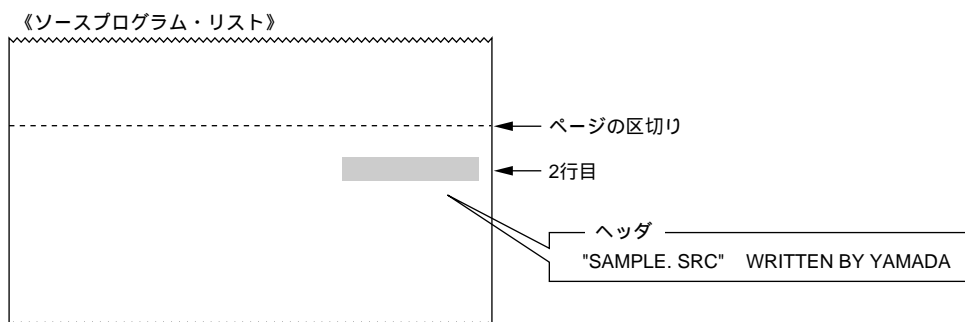
ソースプログラムリストヘッダ設定

.HEADING

書 式 .HEADING "<文字列>"
 ラベルは記述できません。

説 明 ソースプログラムリストのヘッダを設定します。
 ヘッダとして設定できるのは 60 文字以内の文字列です。
 .HEADING は 1 つのソースプログラムの中で何回でも使えます。
 .HEADING による設定の有効範囲は次のようになります。
 ・ ページの 1 行目で設定している場合、そのページから有効
 ・ ページの 2 行目以降で設定している場合、次のページから有効

例 ~
 .HEADING ""SAMPLE.SRC" WRITTEN BY YAMADA"
 ~



ソースプログラムの終わりとエントリポイント指定

.END

書 式 .END [<シンボル>]

説 明 ソースプログラムの終わりを示します。
 .END が出現した時点でアセンブラはアセンブル処理を終了します。
 オペランドに指定したシンボルをエントリポイントとします。
 シンボルには外部定義シンボルを指定します。

例 .EXPORT START
 .SECTION P, CODE, ALIGN=4
START:
 ~
 .END START ;ソースプログラムの終了を宣言しています。
 ;シンボル START がエントリポイントになります。

11.5 ファイルインクルード機能

ファイルインクルードとはアセンブルするソースファイルに他のソースファイルを取り込む機能です(以下、取り込まれる側のソースファイルをインクルードファイルといいます)。

ファイルインクルード機能に関する制御文として.INCLUDE があります。

.INCLUDE を記述した位置に指定のインクルードファイルが取り込まれます。

コーディング例

ソースプログラム

```
.INCLUDE "FILE. H"  
  
.SECTION CD1, CODE, ALIGN = 4  
MOV #ON, R0  
  
~
```

インクルードファイル FILE. H

```
ON: .EQU 1  
OFF: .EQU 0
```

ファイルインクルード結果 (ソースリスト)

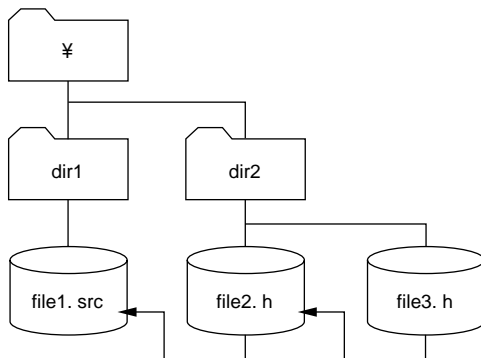
```
.INCLUDE "FILE. H"  
ON: .EQU 1  
OFF: .EQU 0  
  
.SECTION CD1, CODE, ALIGN = 4  
MOV #ON, R0  
  
~
```

.INCLUDE

書 式 .INCLUDE "<ファイル名>"
ラベルは記述できません。

説 明 指定したインクルードファイルを取り込みます。
ファイル名として主ファイル名だけを指定した場合、ファイル型なしのファイル名が有効となります(アセンブラによるファイル型の仮定なし)。
ファイル名はディレクトリを含めた形で指定することができます。
ディレクトリは絶対パス(ルートディレクトリからの経路)または相対パス(カレントディレクトリからの経路)で指定します。
インクルードファイルの中へさらに別のファイルを取り込むこともできます。インクルードは30段階までネストすることができます。
なお、ソースファイル中の.INCLUDEのカレントディレクトリはアセンブラを起動したときのディレクトリとなります。インクルードファイル中の.INCLUDEのカレントディレクトリはそのインクルードファイルが存在するディレクトリとなります。
.INCLUDEで指定したディレクトリ名はincludeオプションで変更することができます。

例 ディレクトリが下図のような構造になっているとき、以下のことを実行するとします。



ルートディレクトリ(¥)からアセンブラを起動

入力ソースファイルは¥dir1¥file1.src

file1.srcにfile2.hをインクルード

file2.hにfile3.hをインクルード

起動コマンドは次のようになります。

```
>asmsh ¥dir1¥file1.src [RET]
```

file1.srcにはつぎのインクルード制御文が必要になります。

```
.INCLUDE "dir2¥file2.h" ; ¥がカレントディレクトリです(相対パス指定)。
```

または

```
.INCLUDE "¥dir2¥file2.h" ; 絶対パス指定
```

file2.hにはつぎのインクルード制御文が必要になります。

```
.INCLUDE "file3.h" ; ¥dir2がカレントディレクトリです(相対パス指定)。
```

または

```
.INCLUDE "¥dir2¥file3.h" ; 絶対パス指定
```

【注】UNIXの場合、円マーク(¥)をスラッシュ(/)に替えてください。

11.6 条件つきアセンブリ機能

11.6.1 条件つきアセンブリ機能の概要

条件つきアセンブリ機能は次のようなアセンブルを簡単に実現します。

- ソースプログラムの文字列を他の文字列に置き換える
- ソースプログラムの一部分をアセンブルするか否か条件によって切り替える
- ソースプログラムの一部分を繰り返し展開してアセンブルする

(1) プリプロセッサ変数

アセンブル条件を記述するための変数をプリプロセッサ変数といいます。プリプロセッサ変数の型には整数型と文字型があります。

(a) 整数型プリプロセッサ変数

.ASSIGNA で定義します(再定義が可能)。プリプロセッサ変数の先頭にバックスラッシュ(¥)とアンパサンド(&)を付けることにより内容を参照できます。

コーディング例

```
FLAG: .ASSIGNA 1
~
.AIF ¥&FLAG EQ 1      ; FLAG が 1 のとき
MOV   R0,R1          ; MOV R0,R1 をアセンブルします。
.AENDI
~
```

(b) 文字型プリプロセッサ変数

.ASSIGNC で定義します(再定義が可能)。プリプロセッサ変数の先頭にバックスラッシュ(¥)とアンパサンド(&)を付けることにより内容を参照できます。

コーディング例

```
FLAG: .ASSIGNC "ON"
~
.AIF "¥&FLAG" EQ "ON" ; FLAG が "ON" のとき
MOV   R0,R1          ; MOV R0,R1 をアセンブルします。
.AENDI
~
```

(2) 置換シンボル

.DEFINE で定義します。ソースプログラムの一部分を指定によって置き換えることができます。コーディングは次のようになります。

コーディング例

```
SYM1: .DEFINE "R1"
~
MOV.L SYM1,R0      ; MOV.L R1,R0 に置き換えられます。
~
```

11. アセンブラ言語仕様

(3) 条件つきアセンブル

ソースプログラムの一部分をアセンブルするか否か条件によって切り替えることができます。

条件つきアセンブリの条件には関係演算子で判別する比較型条件つきアセンブルと置換シンボルで判別する定義型条件つきアセンブルがあります。

(a) 比較型条件つきアセンブル

比較型条件つきアセンブルは条件の成立か不成立かによりアセンブルする範囲を切り替えます。コーディングは次のようになります。

```
~
.AIF      比較型条件
          条件が成立したときアセンブルする部分
.AELIF   比較型条件
          条件が成立したときアセンブルする部分
.AELSE
          全ての条件が成立しないときアセンブルする部分
.AENDI
~
```

この部分は省略可能

コーディング例

```
~
.AIF      "¥&FLAG" EQ "ON"
MOV      R0,R10          ; FLAG が
MOV      R1,R11          ; "ON" のとき
MOV      R2,R12          ; アセンブルします。
.AELSE
MOV      R10,R0          ; FLAG が
MOV      R11,R1          ; "ON" でないとき
MOV      R12,R2          ; アセンブルします。
.AENDI
~
```

(b) 定義型条件つきアセンブル

定義型条件つきアセンブルは置換シンボルが定義されているか否かによりアセンブルする範囲を切り替えます。コーディングは次のようになります。

```
~
.AIFDEF   定義型条件
          条件の置換シンボルが定義されているとき
          アセンブルする部分
.AELSE
          条件の置換シンボルが定義されていないとき
          アセンブルする部分
.AENDI
~
```

この部分は省略可能

コーディング例

```

~
.AIFDEF FLAG
MOV R0,R10 ; .AIFDEFで参照するより前に
MOV R1,R11 ; FLAGが.DEFINEで定義されているとき
MOV R2,R12 ; アセンブルします。
.AELSE
MOV R10,R0 ; .AIFDEFで参照するより前に
MOV R11,R1 ; FLAGが.DEFINEで定義されていないとき
MOV R12,R2 ; アセンブルします。
.AENDI
~

```

(4) 繰り返し展開

ソースプログラムの一部分を指定の回数だけ繰り返し展開してアセンブルすることができます。コーディングは次のようになります。

```

~
.AREPEAT 繰り返し回数
    繰り返しの対象となる部分
.AENDR
~

```

コーディング例

```

; 64ビット÷32ビットの除算を例に挙げます。
; R1:R2(64ビット)÷R0(32ビット)=R2(32ビット):符号無し
TST R0,R0 ; ゼロ除算チェック
BT zero_div
CMP/HS R0,R1 ; オーバフローチェック
BT over_div
DIV0U ; フラグの初期化
.AREPEAT 32
ROTCL R2 ; 32回繰り返し展開してアセンブルします。
DIV1 R0,R1 ;
.AENDR
ROTCL R2 ; R2=商

```

(5) 条件つき繰り返し展開

ソースプログラムの一部分を条件が成立している間、繰り返し展開してアセンブルすることができます。コーディングは次のようになります。

```

~
.AWHILE 繰り返し条件
    繰り返しの対象となる部分
.AENDW
~

```

コーディング例

; 積和演算を例に挙げます。

```
TblSiz: .ASSIGNA 50          ; TblSiz: データテーブルの大きさ
      MOV   A_Tbl1,R1       ; R1: データテーブル 1 の先頭アドレス
      MOV   A_Tbl2,R2       ; R2: データテーブル 2 の先頭アドレス
      CLRMAC                 ; MAC レジスタの初期化
      .AWHILE ¥&TblSiz GT 0 ; TblSiz が 0 より大きい間、
      MAC.W @R1+,@R2+       ; 積和演算を繰り返してアセンブルします。
TblSiz: .ASSIGNA ¥&TblSiz-1 ; TblSiz から 1 を引きます。
      .AENDW
      STS   MACL,R0         ; 結果を R0 に得ます。
```

11.6.2 条件つきアセンブリ機能に関する制御文

条件つきアセンブリ機能の制御文には次のものがあります。

分類	ニーモニック	機能
変数定義に関するもの	.ASSIGNA	整数型プリプロセッサ変数を定義します。再定義が可能です。
	.ASSIGNC	文字型プリプロセッサ変数を定義します。再定義が可能です。
	.DEFINE	プリプロセッサ置換文字列を定義します。再定義できません。
条件分岐に関するもの	.AIF	ソースプログラムの一部分をアセンブルするか否か、条件によって切り替えます。
	.AELIF	
	.AELSE	条件成立の場合は.AIF以降、不成立の場合は.AELIFまたは.AELSE以降のソースプログラムをアセンブルします。
	.AENDI	
	.AIFDEF	ソースプログラムの一部分をアセンブルするか否か、置換シンボルの定義によって切り替えます。
	.AELSE .AENDI	置換シンボルが定義されている場合は.AIFDEF以降、定義されていない場合は.AELSE以降のソースプログラムをアセンブルします。
繰り返し展開に関するもの	.AREPEAT	ソースプログラムの一部分(.AREPEATと.AENDRの間)を指定の回数だけ繰り返して展開してアセンブルします。
	.AENDR	
	.AWHILE	ソースプログラムの一部分(.AWHILEと.AENDWの間)を条件が成立している間、繰り返して展開してアセンブルします。
	.AENDW	
	.EXITM	.AREPEAT、.AWHILEによる繰り返し展開を中断します。
その他	.AERROR	プリプロセッサ展開時のエラー処理をします。
	.ALIMIT	プリプロセッサでの.AWHILEの展開の上限値を設定します。

.ASSIGNA

書 式 <プリプロセッサ変数名>[:] .ASSIGNA <値>

説 明 プリプロセッサ変数を定義します。
プリプロセッサ変数名のつけ方はシンボルの名づけ方と同じです。
また、プリプロセッサ変数名の最大文字数は 32 文字で英大文字と英小文字を区別します。
.ASSIGNA で定義したプリプロセッサ変数は .ASSIGNA によって再定義できます。
プリプロセッサ変数の値は次の形式で指定します。

- ・ 定数 (整数定数、文字定数)
- ・ 既に定義したプリプロセッサ変数
- ・ 上記を項とする式

定義したプリプロセッサ変数は本制御文以降のソースステートメントに対して有効です。
プリプロセッサ変数は以下の箇所で参照できます。

- ・ .ASSIGNA
- ・ .ASSIGNC
- ・ .AIF
- ・ .AELIF
- ・ .AREPEAT
- ・ .AWHILE
- ・ マクロ本体 (.MACRO ~ .ENDM 間のソースステートメント)

プリプロセッサ変数を参照する場合、前にバックスラッシュ (¥) とアンパサンド (&) を付けて記述してください。

¥&プリプロセッサ変数名[']

アポストロフィ (') はプリプロセッサ変数名とソースステートメントの区別を明確にしたい場合に記述します。

オプションでプリプロセッサ文字列が定義されている場合、同名のプリプロセッサ変数に対する .ASSIGNA は無効となります。

例

```

; 汎用多ビットシフト命令を作ります
; SHIFT の値だけ右にシフトします
Rn:      .REG R0                      ;Rn に R0 を設定します。
SHIFT:   .ASSIGNA 27                 ;SHIFT に 27 を設定します。

        .AIF ¥&SHIFT GE 16          ;条件 : SHIFT 16
SHLR16   Rn                          ;条件成立なら Rn を右に 16 ビットシフト。
SHIFT:   .ASSIGNA ¥&SHIFT-16        ;SHIFT から 16 を減じます。
        .AENDI

        .AIF ¥&SHIFT GE 8           ;条件 : SHIFT 8
SHLR8    Rn                          ;条件成立なら Rn を右に 8 ビットシフト。
SHIFT:   .ASSIGNA ¥&SHIFT-8         ;SHIFT から 8 を減じます。
        .AENDI

        .AIF ¥&SHIFT GE 4           ;条件 : SHIFT 4
SHLR2    Rn                          ;条件成立なら Rn を右に 4 ビットシフト。
SHLR2    Rn                          ;
SHIFT:   .ASSIGNA ¥&SHIFT-4         ;SHIFT から 4 を減じます。
        .AENDI

        .AIF ¥&SHIFT GE 2           ;条件 : SHIFT 2
SHLR2    Rn                          ;条件成立なら Rn を右に 2 ビットシフト。
SHIFT:   .ASSIGNA ¥&SHIFT-2         ;SHIFT から 2 を減じます。
        .AENDI

        .AIF ¥&SHIFT EQ 1           ;条件 : SHIFT=1
SHLR     Rn                          ;条件成立なら Rn を右に 1 ビットシフト。
        .AENDI

```

展開結果は次のようになります。

```

SHLR16   R0                      ;条件成立なら Rn を右に 16 ビットシフト。
SHLR8    R0                      ;条件成立なら Rn を右に 8 ビットシフト。
SHLR2    R0                      ;条件成立なら Rn を右に 2 ビットシフト。
SHLR     R0                      ;条件成立なら Rn を右に 1 ビットシフト。

```

.ASSIGNC

書 式 <プリプロセッサ変数名>[:] .ASSIGNC "<文字列>"

説 明 文字型プリプロセッサ変数を定義します。
 プリプロセッサ変数名の付け方はシンボルの名付けかたと同じです。
 また、プリプロセッサ変数名の最大文字数は 32 文字で英大文字と英小文字を区別します。
 .ASSIGNC で定義したプリプロセッサ変数は .ASSIGNC によって再定義できます。
 文字列は文字および既に定義したプリプロセッサ変数をダブルコーテーション(")で囲んで指定します。
 定義したプリプロセッサ変数は本制御文以降のソースステートメントに対して有効です。
 プリプロセッサ変数は以下の箇所で参照できます。

- ・ .ASSIGNA
- ・ .ASSIGNC
- ・ .AIF
- ・ .AELIF
- ・ .AREPEAT
- ・ .AWHILE
- ・ マクロ本体(.MACRO ~ .ENDM 間のソースステートメント)

プリプロセッサ変数を参照する場合、前にバックスラッシュ(¥)とアンパサンド(&)を付けて記述してください。

¥&プリプロセッサ変数名[']

アポストロフィ(')はプリプロセッサ変数名とソースステートメントの区別を明確にしたい場合に記述します。

オプションでプリプロセッサ文字列が定義されている場合、同名のプリプロセッサ変数に対する .ASSIGNC は無効となります。

例

```
FLAG: .ASSIGNC "ON" ; FLAG に "ON" を設定します。
~
.AIF "¥&FLAG" EQ "ON" ; FLAG が "ON" のとき、
MOV R0,R1 ; MOV R0,R1 をアセンブルします。
.AENDI
~
FLAG: .ASSIGNC "¥&FLAG " ; FLAG にスペース(" ")を付加します。
FLAGA: .ASSIGNC "OFF" ; FLAGA に "OFF" を設定します。
FLAG: .ASSIGNC "¥&FLAG'AND ¥&FLAGA"
; FLAG と AND の区別を明確にするため "'" を使います。
; FLAG は結果的に "ON AND OFF" になります。
~
```

プリプロセッサ置換文字列定義

.DEFINE

書 式 <シンボル>[:] .DEFINE "<置換文字列>"

説 明 シンボルの対応する置換文字列に置き換えることを指定します。
 .DEFINE と .ASSIGNC との違いは以下の点です。
 ・ .ASSIGNC で定義したシンボルはプリプロセッサ文でしか使用できませんが、.DEFINE
 で定義したシンボルは任意のステートメントで使用できます。
 ・ .ASSIGNA、.ASSIGNC で定義したシンボルは「¥&シンボル」の形式で参照しますが、
 .DEFINE で定義したシンボルは「シンボル」の形式で参照します。
 .DEFINE で定義したシンボルは再定義できません。
 オプションで置換シンボルが定義されている場合、同名のシンボルに対する .DEFINE は無効
 となります。

例 SYM1: .DEFINE "R1"
 ~
 MOV.L SYM1,R0 ;MOV.L R1,R0 に置き換えられます。
 ~

先頭が a ~ f または A ~ F で始まる 16 進数は .DEFINE で同名のシンボルが定義された場合、
 置換対象になります。置換対象外にするには先頭に 0 を付加してください。

```
C0: .DEFINE "0"
    MOV.B        #H'C0,R0                    ;MOV.B #H'0,R0 に置き換えられます。
    MOV.B        #H'0C0,R0                  ;置き換えられません。
```

基数 (B、Q、D、H) は .DEFINE で同名のシンボルが定義された場合、置換対象になります。
 B、Q、D、H、b、q、d、h 一文字のシンボルを定義するときは注意してください。

```
B: .DEFINE "H"
    MOV.B        #B'10,R0                    ;MOV.H #H'10,R0 に置き換えられます。
```

.AIF, .AELIF, .AELSE, .AENDI

```

書 式      .AIF <項 1> <関係演算子> <項 2>
           <.AIF の条件成立時にアセンブルするソースステートメント>
[ .AELIF <項 1> <関係演算子> <項 2>
  <.AELIF の条件成立時にアセンブルするソースステートメント>]
[ .AELSE
  <全ての条件不成立時にアセンブルするソースステートメント>]
.AENDI

```

説 明 .AIF ~ .AELIF ~ .AELSE ~ .AENDI 間に記述したソースステートメントのうち、条件が成立した部分をアセンブルします。

.AELIF と .AELSE は省略できます。

.AELIF は .AIF と .AELSE の間ならば繰り返し指定できます。

各オペレーションで指定できるオペランドは次のようになります。

オペレーション	オペランド
.AIF	比較型条件
.AELIF	比較型条件
.AELSE	記述不可能
.AENDI	

<項 1>、<項 2>には値または文字列を記述します。ただし、値と文字列を比較すると常に条件不成立となります。

値は定数またはプリプロセッサ変数で指定します。

文字列は文字またはプリプロセッサ変数をダブルコーテーション(")で囲んで指定します。ダブルコーテーション(")自体を文字として指定する場合はダブルコーテーションを2つ続けて記述("\")します。

関係演算子の条件は以下のとおりです。

関係演算子	条件
EQ	項 1 = 項 2
NE	項 1 ≠ 項 2
GT	項 1 > 項 2
LT	項 1 < 項 2
GE	項 1 ≥ 項 2
LE	項 1 ≤ 項 2

【注】 値は 32 ビット符号つき整数として比較します。

文字列の比較は EQ、NE のみ有効です。

例

```
~
.AIF ¥&TYPE EQ 1
MOV      R0,R3      ;TYPE が
MOV      R1,R4      ; 1 のとき
MOV      R2,R5      ;   アセンブルします。
.AELIF ¥&TYPE EQ 2
MOV      R0,R6      ;TYPE が
MOV      R1,R7      ; 2 のとき
MOV      R2,R8      ;   アセンブルします。
.AELSE
MOV      R0,R9      ;TYPE が
MOV      R1,R10     ; 1 でも 2 でもないとき
MOV      R2,R11     ;   アセンブルします。
.AENDI
~
```

.AIFDEF, .AELSE, .AENDI

```
書 式      .AIFDEF <置換シンボル>
           <置換シンボルが定義されていた時にアセンブルするソースステートメント>
[ .AELSE
  <置換シンボルが定義されていない時にアセンブルするソースステートメント>]
  .AENDI
```

ラベルは記述できません。

説 明 .AIFDEF、.AELSE、.AENDI はアセンブルするか否かを置換シンボルの定義によって切り替えます。 .AELSE は省略できます。各オペレーションで指定できるオペランドは次のようになります。

オペレーション	オペランド
.AIFDEF	定義型条件
.AELSE	記述不可能
.AENDI	

置換シンボルは .DEFINE で定義します。

記述した置換シンボルがオプションで定義されている、または本制御文で参照するより前に定義している場合、条件成立となります。記述した置換シンボルが本制御文で参照した後に定義している、または定義がない場合、条件不成立となります。

例

```
~
.AIFDEF FLAG
MOV     R0,R3           ;FLAG が .DEFINE で定義されて
MOV     R1,R4           ;いるときアセンブルします。
.AELSE
MOV     R0,R6           ;FLAG が .DEFINE で定義されて
MOV     R1,R7           ;いないときアセンブルします。
.AENDI
~
```

繰り返し展開

.AREPEAT, .AENDR

書 式 .AREPEAT <回数>
 <繰り返し展開してアセンブルするソースステートメント>
 .AENDR

ラベルは記述できません。

説 明 指定された回数だけ繰り返し展開してアセンブルします。
 各オペレーションで指定できるオペランドは次のようになります。

オペレーション	オペランド
.AREPEAT	繰り返し展開する回数
.AENDR	記述不可能

.AREPEATで指定された回数だけ .AREPEAT ~ .AENDR の間に記述したソースステートメントを繰り返し展開してアセンブルします (ソースステートメントを繰り返しコピーするのと同じで実行時のループにはなりません)。

回数は定数またはプリプロセッサ変数で指定します。

回数に 0 以下の値を指定した場合は展開しません。

例 ; 64 ビット ÷ 32 ビットの除算を例に挙げます。
 ; R1:R2(64 ビット) ÷ R0(32 ビット) = R2(32 ビット) : 符号無し
 TST R0,R0 ; ゼロ除算チェック
 BT zero_div
 CMP/HS R0,R1 ; オーバフローチェック
 BT over_div
 DIV0U ; フラグの初期化
 .AREPEAT 32
 ROTCL R2 ; 32 回繰り返してアセンブルします。
 DIV1 R0,R1 ;
 .AENDR
 ROTCL R2 ; R2=商

.AWHILE, .AENDW

書 式 .AWHILE <項 1> <関係演算子> <項 2>
 <繰り返し展開してアセンブルするソースステートメント>
 .AENDW
 ラベルは記述できません。

説 明 条件が成立している間だけ繰り返し展開してアセンブルします。
 .AWHILE で指定した条件が成立している間、.AWHILE ~ .AENDW の間に記述したソースステートメントを繰り返し展開してアセンブルします(ソースステートメントを繰り返しコピーするのと同じで実行時のループにはなりません)。
 <項 1>、<項 2>は値または文字列を記述します。ただし、値と文字列を比較すると常に条件不成立となります。
 値は定数またはプリプロセッサ変数で指定します。
 文字列は文字またはプリプロセッサ変数をダブルコーテーション(")で囲んで指定します。ダブルコーテーション(")自体を文字として指定する場合は、ダブルコーテーションを2つ続けて記述("")します。
 条件つき繰り返し展開は最終的に条件を不成立にして展開を終了します。
 条件が不成立にならない場合は 65,535 回または .ALIMIT で指定した展開回数を繰り返します。条件の指定にはよく注意してください。
 関係演算子の条件は以下のとおりです。

関係演算子	条件
EQ	項 1 = 項 2
NE	項 1 ≠ 項 2
GT	項 1 > 項 2
LT	項 1 < 項 2
GE	項 1 ≥ 項 2
LE	項 1 ≤ 項 2

【注】 値は 32 ビット符号つき整数として比較します。
文字列の比較は EQ、NE のみ有効です。

例 ; 積和演算を例に挙げます。
TblSiz: .ASSIGNA 50 ;TblSiz: データテーブルの大きさ
 MOV A_Tbl1,R1 ;R1: データテーブル1の先頭アドレス
 MOV A_Tbl2,R2 ;R2: データテーブル2の先頭アドレス
 CLRMAC ;MAC レジスタの初期化
 .AWHILE %&TblSiz GT 0 ;TblSiz が 0 より大きい間、
 MAC.W @R0+,@R1+ ;積和演算を繰り返してアセンブルします。
TblSiz: .ASSIGNA %&TblSiz-1 ;TblSiz から 1 を引きます。
 .AENDW
 STS MACL,R0 ;結果を R0 に得ます。

.EXITM

書 式	.EXITM ラベルは記述できません。
説 明	繰り返し展開(.AREPEAT ~ .AENDR)および条件つき繰り返し展開(.AWHILE ~ .AENDW)の展開を中断させます。 各展開では本制御文が出現した時点で展開を中断します。 本制御文はマクロ展開の中断終了にも使用します。マクロ命令と繰り返し展開を組み合わせで使用する場合は本制御文の位置に注意してください。
例	<pre> ~ COUNT: .ASSIGNA 0 ;COUNT に 0 を設定しています。 .AWHILE 1 EQ 1 ;無限展開 (常に条件成立) を指定しています。 ADD R0,R1 ADD R2,R3 COUNT: .ASSIGNA ¥&COUNT+1 ;COUNT に 1 を加えます。 .AIF ¥&COUNT EQ 2 ;条件は COUNT=2 です。 .EXITM ;条件成立で .AWHILE を中断終了します。 .AENDI .AENDW ~ COUNT が更新され、.AIF の条件が成立すると .EXITM がアセンブルされます。 .EXITM がアセンブルされた時点で .AWHILE の展開を中断終了します。 展開結果は以下のようになります。 ADD R0,R1 ... COUNT が 0 のとき ADD R2,R3 ADD R0,R1 ... COUNT が 1 のとき ADD R2,R3 この後、COUNT は 2 となり、展開は中断終了します。 </pre>

.AERROR

書 式	<code>.AERROR</code> ラベルは記述できません。
説 明	<code>.AERROR</code> をアセンブルするとエラー667 を発生し、アセンブラをエラー終了します。 <code>.AERROR</code> はプリプロセッサ変数の値のチェック等に使用することができます。
例	<pre> ~ .AIF ¥&FLG EQ 1 MOV R1,R10 MOV R2,R11 .AELSE .AERROR ;¥&FLG が 1 以外の場合エラーとします。 .AENDI ~ </pre>

.ALIMIT

書 式	<code>.ALIMIT <回数></code> ラベルは記述できません。
説 明	<p>条件つき繰り返し展開(<code>.AWHILE</code> ~ <code>.AENDW</code>)で、ステートメントの展開回数の上限值を設定します。</p> <p><回数>の値は次の形式で指定します。</p> <ul style="list-style-type: none"> ・定数(整数定数、文字定数) ・既に定義したプリプロセッサ変数 ・上記を項とする式 <p><code>.ALIMIT</code> で指定した上限値を越えるとウォーニング 854 となり、展開を打ち切ります。</p> <p>展開回数の限界値は <code>.ALIMIT</code> を指定しないとき、65,535 です。</p> <p>繰り返し展開回数の上限值は、本制御命令で再指定することで値を変更できます。</p> <p>上限値の再指定は、本制御命令以降のソースステートメントに対して有効です。</p>
例	<pre> .ALIMIT 20 ~ FLG: .ASSIGNA 0 .AWHILE ¥&FLG EQ 0 ;20 回展開した後、展開を打ち切り NOP ;ウォーニングとします。 .AENDW ~ </pre>

11.7 マクロ機能

11.7.1 マクロ機能の概要

本アセンブリ言語ではプログラム中でよく使用する一連の処理に名前をつけ、1つの命令(マクロ命令)として定義することができます。このような定義をマクロ定義といいます。マクロ定義の方法は次のとおりです。

```

~
.MACRO   マクロ名
      マクロ本体
.ENDM
~

```

マクロ名はマクロ命令につける名前、マクロ本体はマクロ命令の内容です。

定義したマクロ命令を呼び出して使用することをマクロコールといいます。マクロコールの方法は次のとおりです。

```

~
定義済みのマクロ名
~

```

マクロ定義とマクロコールの例を以下に示します。

コーディング例

```

~
.MACRO SUM ; R0,R1,R2,R3の合計を求める処理を
MOV R0,R10 ; マクロ命令SUMとして定義します。
ADD R1,R10
ADD R2,R10
ADD R3,R10
.ENDM
~

SUM ; マクロ命令SUMを呼び出します。
; マクロ本体
; MOV R0,R10
; ADD R1,R10
; ADD R2,R10
; ADD R3,R10
; が展開されます。

```

定義したマクロ命令を一部変更して展開することも可能です。手順は次のとおりです。

- (1) マクロ定義
 .MACRO文で仮引数を定義(マクロ名につづいて記述)します。
 マクロ本体の記述に仮引数を使います(仮引数の先頭にバックスラッシュ(¥)を付けます)。
- (2) マクロコール
 マクロパラメータを付けてマクロ命令を呼出します。

マクロ命令展開の際、仮引数是对应するマクロパラメータに置き換えられます。

コーディング例

```
~
.MACRO SUM ARG1          ; 仮引数 ARG1 を定義します。
MOV   R0, ¥ARG1         ; ARG1 を使ってマクロ本体を記述しています。
ADD   R1, ¥ARG1
ADD   R2, ¥ARG1
ADD   R3, ¥ARG1
.ENDM
~
SUM   R10                ; マクロパラメータ R10 を付けてマクロ命令 SUM を呼び出します。
                        ; マクロ本体中の仮引数がマクロパラメータで置き換えられ、
                        ;     MOV   R0, R10
                        ;     ADD   R1, R10
                        ;     ADD   R2, R10
                        ;     ADD   R3, R10
                        ; が展開されます。
```

11.7.2 マクロ機能に関する制御文

マクロ機能の制御文には次のものがあります。

ニーモニック	機能
.MACRO	マクロ命令を定義します。
.ENDM	
.EXITM	マクロ命令の展開を中断します。 11.6.2 .EXITM を参照してください。

.MACRO, .ENDM

書 式 .MACRO <マクロ名>[<仮引数>[,...]]
 .ENDM
 <仮引数> : <仮引数名> [= <仮引数のデフォルト>]
 ラベルは記述できません。

説 明 マクロ定義(一連のソースステートメントに名前を付けてまとめて扱うこと)を行います。
 .MACRO ~ .ENDM 間のソースステートメント(マクロ本体)をマクロ命令として名前を付ける
 ことをマクロ命令を定義する(マクロ定義)といいます。
 各オペレーションで指定できるオペランドは次のようになります。

オペレーション	オペランド
.MACRO	・マクロ命令 ・仮引数 デフォルトを記述(省略可)
.ENDM	記述不可能

(1) マクロ名

マクロ名はマクロ命令に付ける名前です。
 マクロ命令を展開する際にマクロ本体の一部を置換して展開したい場合に指定します。
 仮引数はマクロ展開を行なう(マクロコール)時に指定された文字列(マクロパラメータ)に置換されます。
 マクロ本体では置換したい部分に仮引数名を記述します。マクロ本体での仮引数の参照方法は次のとおりです。
 ¥仮引数名[']
 アポストロフィー(')は、仮引数名とソースステートメントの区別を明確にしたい場合に記述します。

(2) 仮引数

仮引数には仮引数のデフォルトを設定できます。仮引数のデフォルトにはマクロコール時にマクロパラメータを省略した場合に置換する文字列を指定します。
 仮引数の書き方はシンボル名の書き方と同じです。
 仮引数名の最大文字数は 32 文字で英大文字と英小文字を区別します。

(3) 仮引数のデフォルト

仮引数のデフォルトに次の文字を含む場合は文字列をダブルコーテーション(")またはアングルブラケット(<>)で囲んでください。

- ・スペース
- ・タブ
- ・カンマ(,)
- ・セミコロン(;)
- ・ダブルコーテーション(")
- ・アングルブラケット(<>)

マクロ展開では文字列を囲んだダブルコーテーション(")やアングルブラケット(<>)は取り除いて置換します。

(4) 制限

マクロ命令は次の場所では定義できません。

- ・マクロ本体 (.MACRO ~ .ENDM)
- ・.AREPEAT ~ .AENDR の間
- ・.AWHILE ~ .AENDW の間

マクロ本体には .END を記述できません。

.ENDM のラベルにはシンボルを記述できません。

.ENDM のラベルにシンボルを記述した場合は .ENDM を無視します。この場合、エラーは表示しません。

例

```

~
.MACRO SUM ;R0,R1,R2,R3 の合計を求める処理を
MOV R0,R10 ;マクロ命令 SUM として定義します。
ADD R1,R10
ADD R2,R10
ADD R3,R10
.ENDM
~
SUM ;マクロ命令 SUM を呼び出します。
;マクロ本体
; MOV R0,R10
; ADD R1,R10
; ADD R2,R10
; ADD R3,R10
;が展開されます。

```

11.7.3 マクロ本体

.MACRO と .ENDM の間に記述した一連のソースステートメントをマクロ本体と呼びます。マクロ本体はマクロコール(マクロ命令を呼び出すこと)により、展開してアセンブルされます。本節ではマクロ本体が持つ機能と記述方法を説明します。

(1) 仮引数の参照

マクロ展開でマクロパラメータと置換したい部分に仮引数を記述します。仮引数の参照方法は以下のとおりです。

¥仮引数 []

アポストロフィ(')は仮引数名とソースステートメントの区別を明確にしたい場合に記述します。

コーディング例

```
.MACRO PLUS1 P,P1      ; P,P1 は仮引数です。
ADD #1,¥P1            ; 仮引数 P1 を参照しています。
.SDATA "¥P'1"        ; 仮引数 P を参照しています。
.ENDM
PLUS1 R,R1           ; PLUS1 を展開します。
```

展開結果は次のようになります。

```
ADD #1,R1            ; 仮引数 P1 を参照しています。
.SDATA "R1"         ; 仮引数 P を参照しています。
```

(2) プリプロセッサ変数の参照

マクロ本体ではプリプロセッサ変数を参照できます。プリプロセッサ変数の参照方法は次のとおりです。

¥&プリプロセッサ変数名 []

アポストロフィ(')はプリプロセッサ変数とソースステートメントの区別を明確にしたい場合に記述します。

コーディング例

```
.MACRO PLUS1
ADD #1,R¥&V1          ; プリプロセッサ変数 V1 を参照しています。
.SDATA "¥&V'1"       ; プリプロセッサ変数 V を参照しています。
.ENDM
V: .ASSIGNC "R"       ; プリプロセッサ変数 V を定義しています。
V1: .ASSIGNA 1        ; プリプロセッサ変数 V1 を定義しています。
PLUS1                ; PLUS1 を展開します。
```

展開結果は次のようになります。

```
ADD #1,R1            ; プリプロセッサ変数 V1 を参照しています。
.SDATA "R1"         ; プリプロセッサ変数 V を参照しています。
```


(3) マクロ生成番号

マクロ本体にラベルがある場合などは、複数回マクロコールをするとシンボル名が重複してしまいます。このような事態を回避するためにはマクロ生成番号を使用してください。マクロ生成番号はマクロ展開で固有の 5 桁の 10 進数(00000 ~ 99999)を展開します。マクロ生成番号は次のように記述してください。

```
¥@
```

シンボル名の一部としてマクロ生成番号を記述しておく、マクロコールのたびに固有のシンボル名となり、重複を避けることができます。1 つのマクロ本体に 2 つ以上のマクロ生成番号を記述できますが、1 回のマクロコールでは同じマクロ生成番号が展開されます。また、マクロ生成番号は数字に展開されるのでシンボル名の先頭には記述しないでください。

コーディング例

```
.MACRO RES_STR STR,Rn
MOV.L #str¥@,¥Rn
BRA end_str¥@
NOP
str¥@ .SDATA "¥STR"
.ALIGN 2
end_str¥@
.ENDM
RES_STR "ONE",R0 ; RES_STR を展開するたびに
RES_STR "TWO",R1 ; 異なるシンボルを生成します。
```

展開結果は次のようになります。

```
MOV.L #str00000,R0
BRA end_str00000
NOP
str00000 .SDATA "ONE"
.ALIGN 2
end_str00000
MOV.L #str00001,R1
BRA end_str00001
NOP
str00001 .SDATA "TWO"
.ALIGN 2
end_str00001
```

(4) マクロ処理除外

マクロ本体内にバックスラッシュ(¥)があるとマクロ置換処理の対象になります。したがって、バックスラッシュ(¥)を ASCII 文字として記述したい場合はマクロ置換処理から除外する必要があります。マクロ処理除外の書き方は次のとおりです。

```
¥(マクロ処理除外文字列)
```

マクロ展開ではバックスラッシュ(¥)とカッコは取り除きます。

11. アセンブラ言語仕様

コーディング例

```
.MACRO BACK_SLASH_SET
    ¥(MOV    #"¥",R0)          ; ¥は ASCII 文字として展開されます。
.ENDM
```

展開結果は次のようになります。

```
MOV    #"¥",R0              ; ¥は ASCII 文字として展開されます。
```

(5) マクロ内コメント

マクロ本体のコメントをマクロ展開では展開したくない(リスティングファイルに同じコメントが何度も現れるのを避けたい)場合にマクロ内コメントを記述します。マクロ内コメントの書き方は次のとおりです。

¥;コメント

コーディング例

```
.MACRO PUSH Rn
    MOV.L  ¥Rn,@-R15          ; ¥; ¥Rn はレジスタです。
.ENDM
PUSH    R0
```

展開結果は次のようになります(コメントは展開されません)。

```
MOV.L  R0,@-R15
```

(6) 文字列操作関数

マクロ本体には文字列操作関数を記述できます。文字列操作関数には次のものがあります。

- .LEN 関数：文字列の長さ
- .INSTR 関数：文字列の検索
- .SUBSTR 関数：文字列の切り出し

11.7.4 マクロコール

マクロ定義により定義されたマクロ命令を展開することをマクロコールといいます。マクロコールの書き方は次のとおりです。

```
[<シンボル>[:]] <マクロ名> [ <マクロパラメータ>[,...]]
<マクロパラメータ>[= <仮引数名>] = <文字列>
```

マクロ名は、マクロコールする以前にマクロ定義(.MACRO)します。
マクロパラメータには、マクロ展開で置換する文字列を指定します。
この場合、マクロ名に対応するマクロ定義(.MACRO)で、仮引数を宣言しておく必要があります。

(1) マクロパラメータの指定方法

マクロパラメータの指定方法には、位置指定とキーワード指定があります。

(2) 位置指定

マクロ定義(.MACRO)で宣言した仮引数の並び順と、マクロパラメータの並び順を一致させて指定する方法です。

(3) キーワード指定

マクロ定義(.MACRO)で宣言した仮引数の仮引数名にイコール(=)で区切って指定する方法です。

(4) マクロパラメータの書き方

マクロパラメータに次の文字を含む場合は、文字列をダブルコーテーション(")または、アングルブラケット(<>)で囲んでください。

- ・スペース
- ・タブ
- ・カンマ(,)
- ・セミコロン(;)
- ・ダブルコーテーション(")
- ・アングルブラケット(<>)

マクロ展開では、文字列を囲んだダブルコーテーションや、アングルブラケットは取り除いて置換します。

11. アセンブラ言語仕様

コーディング例

```
.MACRO SUM FROM=0,TO=9 ; マクロ命令 SUM、仮引数 FROM,TO を定義します。
MOV R¥FROM,R10
COUNT .ASSIGNA ¥FROM+1
.AWHILE ¥&COUNT LE ¥TO
MOV R¥&COUNT,R10
COUNT .ASSIGNA ¥&COUNT+1
.AENDW
.ENDM
```

仮引数を用いてマクロ本体を記述しています。

```
SUM 0,5
SUM TO=5
```

どちらも同じ展開結果になります。

マクロ本体中の仮引数がマクロパラメータで置き換えられ、展開結果は次のようになります。

```
MOV R0,R10
MOV R1,R10
MOV R2,R10
MOV R3,R10
MOV R4,R10
MOV R5,R10
```

11.7.5 文字列操作関数

マクロ本体で使用できる文字列操作関数には次のものがあります。

文字列操作関数	機能
.LEN	文字列の長さを返します。
.INSTR	文字列の検索を行ないます。
.SUBSTR	文字列の切り出しを行ないます。

文字列の長さ

.LEN

書 式 .LEN[]("<文字列>")

説 明 文字列の長さを数え、基数を省略した 10 進数に置換します。
文字列は文字をダブルコーテーション(")で囲んで指定します。
ダブルコーテーション自体を文字として指定する場合は 2 つ続けて記述します。
文字列にはマクロの仮引数、プリプロセッサ変数を指定できます。

```
.LEN("¥仮引数名")
.LEN("¥&プリプロセッサ変数名")
```

本関数を記述できるのはマクロ本体(.MACRO ~ .ENDM)だけです。

例

```
~
.MACRO RESERVE_LENGTH P1
.ALIGN 4
.SRES        .LEN("¥P1")
.ENDM
~
RESERVE_LENGTH ABCDEF
RESERVE_LENGTH ABC
```

展開結果は次のようになります。

```
.ALIGN 4
.SRES        6     ; "ABCDEF"の字数は 6 です。
.ALIGN 4
.SRES        3     ; "ABC"の字数は 3 です。
```


文字列の切り出し

.SUBSTR

書 式 .SUBSTR[] ("<文字列>", <切り出しの開始位置>, <切り出しの長さ>)

説 明 文字列の先頭を 0 とした切り出しの開始位置から、切り出しの長さ分の文字列を切り出し、ダブルコーテーション (") で囲んだ文字列に置換します。
文字列は文字をダブルコーテーションで囲んで指定します。
ダブルコーテーション自体を文字として指定する場合は 2 つ続けて記述します。
切り出しの開始位置は 0 以上が指定できます。
切り出しの長さは 1 以上が指定できます。
切り出しの開始位置、切り出しの長さが不適当な場合には空文字 (" ") に置換します。
切り出しの開始位置、切り出しの長さにはマクロの仮引数、プリプロセッサ変数を指定できます。

```
.SUBSTR ("¥仮引数名", . . . .)
```

```
.SUBSTR ("¥&プリプロセッサ変数名", . . . .)
```

本関数を記述できるのはマクロ本体 (.MACRO ~ .ENDM) だけです。

例

```
~
.MACRO RESERVE_STR P1=0,P2
.SDATA      .SUBSTR("ABCDEFGH",¥P1,¥P2)
.ENDM
~
RESERVE_STR 2,2
RESERVE_STR ,3 ;マクロパラメータ P1 を省略しています。
```

展開結果はそれぞれ次のようになります。

```
.SDATA      "CD"
.SDATA      "ABC"
```

11.8 リテラルプール自動生成機能

11.8.1 リテラルプール自動生成機能の概要

2 バイト長、4 バイト長の定数データ(以下、リテラルといいます)をレジスタへ転送するには、リテラルプール(リテラルの集まり)を確保し、PC 相対アドレス形式で参照しなければなりません。リテラルプールを配置する場合、次のような考慮をする必要があります。

- データ転送命令が参照できる範囲にデータが位置しているか？
- 2 バイト長のデータは2 バイト境界、4 バイト長のデータは4 バイト境界に位置しているか？
- 1 つのデータを複数のデータ転送命令で共有できないか？
- プログラム中のどこにリテラルプールを配置するか？

リテラルプール自動生成機能とは定数データのレジスタ転送に対応する PC 相対の MOV 命令(または MOVA 命令)と .DATA を 1 つの命令から自動的に生成する機能です。例えば、以下のプログラム(a)は本機能を用いれば(b)のように記述できます

プログラム(a)

```

MOV.L      DATA1,R0
MOV.L      DATA2,R1
~
.ALIGN     4
DATA1     .DATA.L  H'12345678
DATA2     .DATA.L  500000

```

プログラム(b)

```

MOV.L      #H'12345678,R0
MOV.L      #500000,R1
~

```

11.8.2 リテラルプール自動生成機能に関する拡張命令

拡張命令(MOV.W #imm,Rn、MOV.L #imm,Rn、MOVA #imm,R0)の記述に対してアセンブラは必要なリテラルプールを自動生成し、PC 相対のディスプレースメント値を計算します。

表 11.28 に拡張命令のソースステートメントとその展開結果を示します。拡張命令のソースステートメントは 1 つの実行命令と 1 つのリテラルデータとに展開されます。

表 11.28 拡張命令の種類と展開結果

拡張命令	展開結果
MOV.W #imm,Rn	MOV.W @(disp,PC),Rn と 2 バイト長のリテラルデータ
MOV.L #imm,Rn	MOV.L @(disp,PC),Rn と 4 バイト長のリテラルデータ
MOVA #imm,R0	MOVA @(disp,PC),R0 と 4 バイト長のリテラルデータ

11.8.3 リテラルプール自動生成機能のサイズモード

リテラルプール自動生成には、オペレーションサイズを指定したデータ転送命令(拡張命令)によりリテラルプールを自動生成するサイズ指定モード、オペレーションサイズの指定がないデータ転送命令をアセンブラが imm の値により適切な命令を選択するサイズ選択モードの 2 つのサイズモードがあります。

表 11.29 にデータ転送命令とサイズモードの関係を示します。

表 11.29 データ転送命令とサイズモードの関係

データ転送命令	サイズ指定モード	サイズ選択モード
MOV #imm,Rn	実行命令	アセンブラが選択
MOV.B #imm,Rn	実行命令	実行命令
MOV.W #imm,Rn	拡張命令	拡張命令
MOV.L #imm,Rn	拡張命令	拡張命令

(1) サイズ指定モード

サイズ指定モードではオペレーションサイズ指定がないデータ転送命令(MOV #imm,Rn)は通常の実行命令として扱われます。サイズ指定モードは auto_literal オプションが指定されていない場合に有効となります。

(2) サイズ選択モード

サイズ選択モードでは拡張命令のほかにオペレーションサイズを指定していないデータ転送命令(MOV #imm,Rn)の記述に対してアセンブラが imm の値の範囲を判定し、必要ならばリテラルプールを自動生成します。imm 値は符号付きの範囲で判定します。サイズ選択モードは auto_literal オプションを指定した場合に有効となります。

表 11.30 にサイズ選択モードで imm の値の範囲により選択される命令を示します。

表 11.30 サイズ選択モードで選択される命令

imm の指定方法	imm の値の範囲*	選択される命令
定数値	H'FFFFFF80 ~ H'0000007F	MOV.B #imm,Rn
参照前に定義された定数シンボル	(-128 ~ 127)	
参照前に定義された絶対アドレスシンボル	H'FFFF8000 ~ H'FFFFFF7F (-32,768 ~ -129)	MOV.W #imm,Rn 展開結果
	H'00000080 ~ H'00007FFF (128 ~ 32,767)	(MOV.W @(disp,PC),Rn と 2 バイト長のリテラルデータ)
	H'80000000 ~ H'FFFF7FFF (-2,147,483,648 ~ -32,769)	MOV.L #imm,Rn 展開結果
	H'00008000 ~ H'7FFFFFFF (32,768 ~ 2,147,483,647)	(MOV.L @(disp,PC),Rn と 4 バイト長のリテラルデータ)
相対アドレスシンボル	imm の値に依存しない	MOV.L #imm,Rn
外部参照シンボル		展開結果
参照後に定義された定数シンボル		(MOV.L @(disp,PC),Rn と
参照後に定義された絶対アドレスシンボル		4 バイト長のリテラルデータ)

【注】 * ()内は 10 進表記

11.8.4 リテラルプールの出力

リテラルプールが出力されるポイントは次のどちらかです。

- 無条件分岐とそのディレイスロット命令に続く位置
- プログラムが .POOL を記述した位置

なお、出力ポイントは literal オプションで選択することができます。

アセンブラは拡張命令の記述位置以降で最も近い出力ポイントに対応するリテラルを出力します。アセンブラは同じポイントに出力するリテラルデータをまとめて 1 つのリテラルプールを生成します。

【注】ディレイスロット命令にラベルが指定されている場合、リテラルプール出力ポイントにはなりません。

- (1) 無条件分岐を利用したリテラルプール出力
出力例を以下に示します。

コーディング例

ソースプログラム

```
. SECTION CD1, CODE, LOCATE = H' 0000F000
CD1_START:
MOV. L  #H' FFFF0000, R0
MOV. W  #H' FF00, R1
MOV. L  #CD1_START, R2
MOV     #H'FF, R3
RTS
MOV     R0, R10
.END
```

リテラルプール自動生成結果 (ソースリスト)

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000	2	CD1_START
3	0000F000 D003	3	MOV. L #H' FFFF0000, R0
4	0000F002 9103	4	MOV. W #H' FF00, R1
5	0000F004 D203	5	MOV. L #CD1_START, R2
6	0000F006 E3FF	6	MOV #H'FF, R3
7	0000F008 000B	7	RTS
8	0000F00A 6A03	8	MOV R0, R10
9			***** BEGIN-POOL *****
10	0000F00C FF00		DATA FOR SOURCE-LINE 4
11	0000F00E 0000		ALIGNMENT CODE
12	0000F010 FFFF0000		DATA FOR SOURCE-LINE 3
13	0000F014 0000F000		DATA FOR SOURCE-LINE 5
14			***** END-POOL *****
15		9	. END

(2) .POOL の位置へのリテラルプールの出力

無条件分岐を利用したリテラルプールの出力がディスプレースメントの範囲内に行えない場合(プログラム中に無条件分岐が少ない場合など)アセンブラはエラー402を出力します。このような場合、.POOLをディスプレースメントの範囲内に記述してください。

ディスプレースメントの範囲は次のとおりです。

- オペレーションサイズがワード(W)のとき：0～511バイト
- オペレーションサイズがロング(L)のとき：0～1023バイト

.POOLの位置へのリテラルプールの出力ではそのリテラルプールの飛び越すための分岐命令も一緒に出力されます。

コーディング例

ソースプログラム

```
. SECTION CD1, CODE, LOCATE = H' 0000F000
CCD1_START
MOV. L  #H' FFFF0000, R0
MOV. W  #H' FF00, R1
MOV. L  #CD1_START, R2
MOV     #H'FF, R3
.POOL
.END
```

リテラルプール自動生成結果 (ソースリスト)

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000	2	CD1_START:
3	0000F000 D003	3	MOV. L #H' FFFF0000, R0
4	0000F002 9103	4	MOV. W #H' FF00, R1
5	0000F004 D203	5	MOV. L #CD1_START, R2
6	0000F006 E3FF	6	MOV #H' FF, R3
7	0000F008	7	. POOL
8			***** BEGIN-POOL *****
9	0000F008 A006		BRA TO END-POOL
10	0000F00A 0009		NOP
11	0000F00C FF00		DATA FOR SOURCE-LINE 4
12	0000F00E 0000		ALIGNMENT CODE
13	0000F010 FFFF0000		DATA FOR SOURCE-LINE 3
14	0000F014 0000F000		DATA FOR SOURCE-LINE 5
15			***** END-POOL *****
16		8	. END

11.8.5 リテラルの共有

アセンブラは同一のリテラルプールに入る複数の拡張命令の imm を共有する処理を行いません。以下に示すもので同一のものを共有します。

- (1) シンボル
- (2) 定数
- (3) シンボル ± 定数

上記のほか、アセンブル時に同一の値を持つと判断できる式は共有する場合があります。imm の値が同じでも拡張命令のオペレーションサイズが異なる場合はリテラルデータを共有しません。また、出力先リテラルプールが異なる場合もデータは共有しません。複数の拡張命令が 1 つのリテラルデータを共有する例を以下に示します。

コーディング例

ソースプログラム

```

SECTION CD1, CODE, LOCATE = H' 0000F000
CD1_START:
MOV. L #H' FFFF0000, R0
MOV. W #H' FF00, R1
MOV. L #H' FFFF0000, R2
MOV #H' FF, R3
RTS
MOV R0, R10
.END

```

リテラルプール自動生成結果 (ソースリスト)

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000	2	CD1_START:
3	0000F000 D003	3	MOV. L #H' FFFF0000, R0
4	0000F002 9103	4	MOV. W #H' FF00, R1
5	0000F004 D202	5	MOV. L #H' FFFF0000, R2
6	0000F006 E3FF	6	MOV #H' FF, R3
7	0000F008 000B	7	RTS
8	0000F00A 6A03	8	MOV R0, R10
9			***** BEGIN-POOL *****
10	0000F00C FF00		DATA FOR SOURCE-LINE 4
11	0000F00E 0000		ALIGNMENT CODE
12	0000F010 FFFF0000		DATA FOR SOURCE-LINE 3, 5
13			***** END-POOL *****
14		9	. END

11.8.6 リテラルプール出力の抑止

プログラム中に無条件分岐が多すぎると次のような問題が生じます。

- 小さなリテラルプールがいくつも出力される。
- リテラルを共有できない。

このような場合には次の方法でリテラルプール出力を抑止してください。

```

~
遅延分岐命令
ディレイスロット命令
.NOPOOL
~

```

コーディング例

ソースプログラム

<pre> CASE1: MOV. L #H' FFFF0000, R0 RTS NOP .NOPOOL CASE2: MOV. L #H' FFFF0000, R0 RTS NOP </pre>	<pre> ----- 拡張命令1 ----- このポイントへの リテラルプール出力はありません。 ----- ----- 拡張命令2 ----- リテラルプールの出力ポイントです。 ----- </pre>
--	--

リテラルプール自動生成結果 (ソースリスト)

<pre> 20 0000F000 21 0000F000 D002 22 0000F002 000B 23 0000F004 0009 24 25 0000F006 26 0000F006 D001 27 0000F008 000B 28 0000F00A 0009 29 30 0000F00C FFFF0000 31 </pre>	<pre> 20 CASE1: 21 MOV. L #H' FFFF0000, R0 22 RTS 23 NOP 24 .NOPOOL 25 CASE2: 26 MOV. L #H' FFFF0000, R0 27 RTS 28 NOP 29 * * * * * BEGIN-POOL * * * * * 30 DATA FOR SOURCE-LINE 21, 26 31 * * * * * END-POOL * * * * * </pre>
--	--

11.8.7 リテラルプール自動生成に関する注意事項

- (1) 拡張命令を記述してエラーとなる場合
ディレイスロット命令に拡張命令を記述できません(エラー151)。
境界調整数2未満の相対アドレスセクションには拡張命令を記述できません(エラー152)。
境界調整数4未満の相対アドレスセクションにはMOV.L #imm,Rn、MOVA #imm,R0 を記述できません(エラー152)。
- (2) .POOLを記述してエラーとなる場合
遅延分岐命令に続いて.POOLを記述できません(エラー522)。
- (3) .NOPOOLを記述してエラーとなる場合
.NOPOOLはディレイスロット命令に続いて記述された場合に有効です。それ以外の位置に記述された場合、エラー521となります。
- (4) 拡張命令を展開した結果、他の実行命令のディスプレースメントが範囲外となる場合
アセンブラはリテラルプールを生成し、ディスプレースメントが範囲外となる命令をエラー402とします。
.NOPOOLを使用するなどしてリテラルプールの出力ポイントを移動するか、エラーとなる命令の位置またはアドレッシングモードを変更してください。
- (5) リテラルプールの出力位置が見つからない場合
拡張命令の位置からみて、次の条件を満足するリテラルプール出力ポイントが見つからない場合
・同ファイル
・同セクション
・拡張命令の記述された位置以降
アセンブラはその拡張命令が存在するセクションの最後にリテラルプールとそれを飛び越すBRA命令(ディレイスロット命令はNOP)を出力し、ウォーニング876を発行します。
- (6) 拡張命令のディスプレースメントが範囲外となる場合
リテラルプールを生成したが、拡張命令からのディスプレースメントが範囲外となる場合、その拡張命令はエラー402となります。
.POOLを用いるなどして、範囲内となる場所にリテラルプールが生成されるようにしてください。
- (7) サイズ指定モードとサイズ選択モードの相違
Ver.2.0のリテラルプール機能はサイズ指定モードのみのサポートでしたが、Ver.3.1以降ではサイズ選択モードを追加しました。Ver.2.0で作成したソースプログラムをVer.3.1以降のサイズ選択モードでアセンブルするとオペレーションサイズ指定のないデータ転送命令においてimm値がH'00000080~H'000000FF(128~255)の範囲で相違がでます。
サイズ指定モードとサイズ選択モードの出力例を次に示します。

コーディング例

ソースプログラム

```

SECTION CD1, CODE, LOCATE = H' 0000F000
MOV. L   #H' FF, R0
MOV. W   #H' FF, R1
MOV. B   #H' FF, R2
MOV      #H' FF, R3
RTS
MOV      R0, R10
.END

```

サイズ指定モードのリテラルプールの自動生成結果 (ソースリスト)

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000 D003	2	MOV. L #H' FF, R0
3	0000F002 9103	3	MOV. W #H' FF, R1
4	0000F004 E2FF	4	MOV. B #H' FF, R2
5	0000F006 E3FF	5	MOV #H' FF, R3
6	0000F008 000B	6	RTS
7	0000F00A 6A03	7	MOV R0, R10
8			***** BEGIN-POOL *****
9	0000F00C 00FF		DATA FOR SOURCE-LINE 3
10	0000F00E 0000		ALIGNMENT CODE
11	0000F010 000000FF		DATA FOR SOURCE-LINE 2
12			***** END-POOL *****
13		8	. END

R3の内容は、H'FFFFFFFとなります。

サイズ選択モードのリテラルプールの自動生成結果 (ソースリスト)

1	0000F000	1	. SECTION CD1, CODE, LOCATE = H' 0000F000
2	0000F000 D003	2	MOV. L #H' FF, R0
3	0000F002 9103	3	MOV. W #H' FF, R1
4	0000F004 E2FF	4	MOV. B #H' FF, R2
5	0000F006 9301	5	MOV #H' FF, R3
6	0000F008 000B	6	RTS
7	0000F00A 6A03	7	MOV R0, R10
8			***** BEGIN-POOL *****
9	0000F00C 00FF		DATA FOR SOURCE-LINE 3, 5
10	0000F00E 0000		ALIGNMENT CODE
11	0000F010 000000FF		DATA FOR SOURCE-LINE 2
12			***** END-POOL *****
13		8	. END

R3の内容は、H'000000FFとなります。

11.9 リピートループ命令自動生成機能

11.9.1 リピートループ命令自動生成機能の概要

SH-DSP では、LDRS 命令と LDRE 命令によってリピート開始アドレスとリピート終了アドレスを RS レジスタと RE レジスタに設定します。そのアドレスの設定値はリピートループ間の命令数により異なります。アドレスを記述するとき、表 11.31 に示すような考慮をする必要があります。

表 11.31 リピートループ間の命令数とアドレスの設定値

命令数	1 命令	2 命令	3 命令	4 命令以上
RS レジスタ	s_addr0+8	s_addr0+6	s_addr0+4	s_addr
RE レジスタ	s_addr0+4	s_addr0+4	s_addr0+4	e_addr3+4

- ・ s_addr0 : リピート開始アドレスの 1 命令前のアドレス
- ・ s_addr : リピート開始アドレス
- ・ e_addr3 : リピート終了アドレスの 3 命令前のアドレス

リピートループ命令自動生成機能とは、リピートループ間の命令数からアドレス値を RS、RE レジスタに転送する PC 相対の LDRS、LDRE 命令とリピート回数を設定する SETRC 命令を 1 つの命令から自動的に生成する機能です。

例えば、以下のプログラム(a)は本機能を用いると(b)のように記述できます。

プログラム(a)

```

LDRS s_addr0+6
LDRE s_addr0+4
SETRC #10
s_addr0:  NOP
          PADD A0,M0,A0      ;リピート開始アドレス
          PCMP x1,M0        ;リピート終了アドレス

```

プログラム(b)

```

REPEAT s_addr,e_addr,#10
NOP
s_addr:  PADD A0,M0,A0      ;リピート開始アドレス
e_addr:  PCMP X1,M0        ;リピート終了アドレス

```


11.9.2 リピートループ命令自動生成機能に関する拡張命令

拡張命令(REPEAT s_label,e_label,#imm、REPEAT s_label,e_label,Rn、REPEAT s_label,e_label)の記述に対してアセンブラは必要な命令を自動生成し、PC 相対のディスプレースメント値を計算します。表 11.32 に拡張命令のソースステートメントとその展開結果を示します。拡張命令のソースステートメントは 2 つまたは 3 つの実行命令に展開されます。

表 11.32 拡張命令の種類と展開結果

拡張命令	展開結果
REPEAT s_label,e_label,#imm	LDRS@(disp,PC)と LDRE@(disp,PC)と SETRC#imm
REPEAT s_label,e_label,Rn	LDRS@(disp,PC)と LDRE@(disp,PC)と SETRC Rn
REPEAT s_label,e_label	LDRS@(disp,PC)と LDRE@(disp,PC)

11.9.3 REPEAT の記述方法

REPEAT の書き方は次のとおりです。

[<シンボル>[:]] REPEAT <開始アドレス>,<終了アドレス>[,<リピート回数>]

- (1) ステートメントの要素
 - (a) <開始アドレス>、<終了アドレス>
リピートループの開始アドレスと終了アドレスをラベルで記述します。
 - (b) <リピート回数>
リピート回数をイミディエイトまたは汎用レジスタで記述します。
- (2) 説明
 - (a) REPEATは開始アドレスから終了アドレスまでの命令をリピートするための実行命令(LDRS、LDRE)を自動的に生成します。
 - (b) リピート回数を指定した場合、SETRCを生成します。リピート回数を省略した場合、SETRCを生成しません。

11.9.4 コーディング例

(1) 基本例(リピートする命令数が 4 命令以上のとき)

```

        REPEAT RptStart,RptEnd,#5
        PCLR Y0
        PCLR A0
RptStart:  MOVX @R4+,X1  MOVY @R6+,Y1
           PADD A0,Y0,Y0  PMULS X1,Y1,A0
        DCT  PCLR A0
           AND  R0,R4
RptEnd:    AND  R0,R6

```

このプログラムは RptStart から RptEnd までの 5 命令を 5 回繰り返し実行します。

展開イメージ

```

        LDRS RptStart
        LDRE RptEnd3+4
        SETRC #5
        PCLR Y0
        PCLR A0
RptStart:  MOVX @R4+,X1  MOVY @R6+,Y1
RptEnd3:   PADD A0,Y0,Y0  PMULS X1,Y1,A0 ;ラベルは実際には生成されません
        DCT  PCLR A0
           AND  R0,R4
RptEnd:    AND  R0,R6

```

(2) リピートする命令数が 1 命令のとき

開始アドレスと終了アドレスを同じラベルで指定してください。

```

        REPEAT Rpt,Rpt,R0
        MOVX @R4+,X1  MOVY @R6+,Y1
Rpt:    PADD A0,Y0,Y0  PMULS X1,Y1,A0  MOVX @R4+,X1  MOVY @R6+,Y1
展開イメージ

```

```

        LDRS RptStart0+8
        LDRE RptStart0+4
        SETRC R0
RptStart0:  MOVX @R4+,X1  MOVY @R6+,Y1 ; ラベルは実際には生成されません
Rpt:        PADD A0,Y0,Y0  PMULS X1,Y1,A0  MOVX @R4+,X1  MOVY @R6+,Y1

```

(3) リピートする命令数が 2 命令のとき

```

        REPEAT RptStart,RptEnd,#10
        PCLR Y0
RptStart:  MOVX @R4+,X1  MOVY @R6+,Y1
RptEnd:    PADD A0,Y0,Y0  PMULS X1,Y1,A0
展開イメージ

```

```

        LDRS RptStart0+6
        LDRE RptStart0+4
        SETRC #10
RptStart0:  PCLR Y0 ; ラベルは実際には生成されません
RptStart:   MOVX @R4+,X1  MOVY @R6+,Y1
RptEnd:     PADD A0,Y0,Y0  PMULS X1,Y1,A0

```

(4) リpeatする命令数が3命令のとき

```

        REPEAT RptStart,RptEnd,R0
        PCLR Y0
RptStart:  MOVX @R4+,X1  MOVY @R6+,Y1
           PMULS X1,Y1,A0
RptEnd:    PADD A0,Y0,Y0
展開イメージ

        LDRE RptStart0+4
        LDRS RptStart0+4
        SETRC R0
RptStart0: PCLR Y0          ; ラベルは実際には生成されません
RptStart:  MOVX @R4+,X1  MOVY @R6+,Y1
           PMULS X1,Y1,A0
RptEnd:    PADD A0,Y0,Y0

```

(5) リpeat回数を省略したとき

リpeat回数を省略するとアセンブラは SETRC を展開しません。LDRS、LDRE と SETRC を分離したいとき使用します。

```

        REPEAT RptStart,RptEnd
        ; この部分に LDRS, LDRE が展開されます
        MOV #10,R0
OuterLoop:
        SETRC #16
        PCLR Y0
        PCLR A0
RptStart:  MOVX @R4+,X1  MOVY @R6+,Y1
           PADD A0,Y0,Y0  PMULS X1,Y1,A0
        DCT  PCLR A0
           AND R0,R4
RptEnd:    AND R0,R6
           DT R0
           BF OuterLoop

```

11.9.5 REPEAT 拡張命令に関する注意事項

(1) 開始アドレス、終了アドレスに関する注意事項

開始アドレスおよび終了アドレスに指定できるラベルは同じセクション内のラベルまたは同じローカルブロック内のローカルラベルです。

開始アドレスは REPEAT 拡張命令より後のアドレスになければなりません。

また、終了アドレスは開始アドレスより後のアドレスになければなりません。

(2) ループ内に記述する命令に関する注意事項

- (a) ループ内にデータまたはデータ領域を確保する制御命令または .ORG を記述した場合、アセンブラはウォーニングを出力し、1 制御命令を 1 命令としてリPEATする命令数をカウントします。 .ALIGN を記述してアライメントが生成された場合、アセンブラはウォーニングを出力し、 .ALIGN を 1 命令としてリPEATする命令数をカウントします。該当する制御命令は次のとおりです。

.DATA、.DATAB、.SDATA、.SDATAB、.SDATAC、.SDATAZ、.FDATA、

.FDATAB、.XDATA、.RES、.SRES、.SRESC、.SRESZ、.FRES、.ALIGN、.ORG

- (b) アセンブラはループ内ではリテラルプールの自動生成を抑制します。したがって、無条件分岐命令があってもリテラルプールの出力対象になりません。また、.POOL を記述した場合、アセンブラはウォーニングを出力し、.POOL を無視します。

(3) ループ直前の命令に関する注意事項

リPEATする命令数が 3 命令以下の場合、ループ直前の命令は実行命令または DSP 命令でなければなりません。したがって、リPEATする命令数が 3 命令以下で開始アドレスの直前が以下の場合、アセンブラはエラーを出力します。

- (a) データまたはデータ領域を確保する制御命令または .ORG

.DATA、.DATAB、.SDATA、.SDATAB、.SDATAC、.SDATAZ、.FDATA、

.FDATAB、.XDATA、.RES、.SRES、.SRESC、.SRESZ、.FRES、.ORG

- (b) リテラルプール自動生成機能で生成されたリテラルプール

開始アドレスの直前が無条件分岐命令+ディレイスロット命令または .POOL の場合、リテラルプールが自動的に生成される可能性があります。リテラルプールの生成を抑制するためにはディレイスロット命令の直後に .NOPOOL を記述してください。

- (c) .ALIGN で 1 バイトのアライメントが生成された場合

.ALIGN を指定したとき、直前が奇数アドレスである場合、1 バイトのアライメントが生成される場合があります(例えば、ロケーションカウンタが 3 で .ALIGN 4 を指定したときなど)。この場合、実行命令でないデータがループの直前に出力されたことになり、エラーになります。2 バイト以上のアライメントが出力された場合、直前の命令は NOP になり、正常に動作します。

(4) その他の注意事項

- (a) REPEAT 拡張命令 ~ 開始アドレスの間には 1 命令以上の実行命令または DSP 命令がなければなりません。1 命令以上の実行命令または DSP 命令が存在しない場合、アセンブラはエラーを出力します。

- (b) REPEAT 拡張命令 ~ 終了アドレスの間に別の REPEAT 拡張命令を記述することはできません。REPEAT 拡張命令をネストして記述した場合、アセンブラはエラーを出力し、別の REPEAT を無視します。

12. コンパイラのエラーメッセージ

12.1 エラー形式とエラーレベル

本章では、以下の形式で出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ
エラー内容

エラーレベルは、エラーの重要度に従い、5種類に分類されます。

エラーレベル	動作
(I) インフォメーション	処理を継続します。
(W) ウォーニング	処理を継続します。
(E) エラー	オプション解析処理を継続し、処理を中断します。
(F) フェータル	処理を中断します。
(-) インターナル	処理を中断します。

12.2 メッセージ一覧

C0001 (I) "文字列" in comment
注釈の中に、"文字列"があります。

C0002 (I) No declarator
宣言子のない宣言があります。

C0003 (I) Unreachable statement
実行されることのない文があります。

C0004 (I) Constant as condition
if 文または switch 文の条件を示す式として、定数式を指定しています。

C0005 (I) Precision lost
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。

C0006 (I) Conversion in argument
関数の引数の式が、原型宣言で指定した引数の型に変換されます。

C0008 (I) Conversion in return
リターン文の式が、関数の返す値の型に変換されます。

12. コンパイラのエラーメッセージ

- C0010 (I) Elimination of needless expression
不要な式があります。
- C0011 (I) Used before set symbol : "変数名"
値の設定されていない局所変数を参照しています。
- C0012 (I) Unused variable "変数名"
使用していない変数があります。
- C0015 (I) No return value
void 型以外の型を返す関数の中で、リターン文が値を返していないか、またはリターン文がありません。
- C0100 (I) Function "関数名" not optimized
関数のサイズが大きすぎるため、最適化できません。
- C0200 (I) No prototype function
関数の原型宣言がありません。
- C1000 (W) Illegal pointer assignment
ポインタ型どうしの代入で、それぞれのポインタ型の指す型が異なります。
- C1001 (W) Illegal comparison in "演算子"
二項演算子==または!=の被演算子が、一方がポインタ型で他方が値 0 以外の汎整数型を指しています。
- C1002 (W) Illegal pointer for "演算子"
二項演算子=、!=、>、<、>=または<=の被演算子が、同じ型へのポインタ型を指していません。
- C1005 (W) Undefined escape sequence
文字定数または文字列の中で、文法上定義していない拡張表記(バックスラッシュとそれに続く文字)を用いています。
- C1007 (W) Long character constant
文字定数の長さが 2 文字以上になっています。
- C1008 (W) Identifier too long
識別子の長さが 8189 文字を超えています。8190 文字以降は無効となります。
- C1010 (W) Character constant too long
文字定数の長さが 4 文字を超えています。
- C1012 (W) Floating point constant overflow
浮動小数点定数の値が値の範囲を超えています。符号にしたがって+ または- に対応する内部表現の値を仮定します。

- C1013 (W) Integer constant overflow
整数定数の値が unsigned long 型のとり得る値の範囲を超えています。オーバーフローした上位ビットを無視した値を仮定します。
- C1014 (W) Escape sequence overflow
文字定数あるいは文字列の中でのビットパターンを示す拡張表記の値が 255 を超えています。下位 1 バイトの値を有効とします。
- C1015 (W) Floating point constant underflow
浮動小数点定数の値の絶対値が表現できる最小値よりも小さな値となっています。定数の値を 0.0 と仮定します。
- C1016 (W) Argument mismatch
原型宣言の中の引数と関数呼び出しの対応する引数の型がポインタ型で、それぞれの指す型が異なっています。関数呼び出しにおける引数のポインタの内部表現をそのまま設定します。
- C1017 (W) Return type mismatch
関数の返す型とリターン文の式の型がポインタ型で、それぞれの指す型が異なっています。リターン文の式のポインタの内部表現をそのまま設定します。
- C1019 (W) Illegal constant expression
定数式において関係演算子<、>、<=または>=の被演算子が、同じ型へのポインタ型を指していません。結果の値を 0 と仮定します。
- C1020 (W) Illegal constant expression of "--"
定数式において二項演算子-の被演算子が、同じ型へのポインタ型を指していません。結果の値を 0 と仮定します。
- C1021 (W) Register saving pragma conflicts in interrupt function "関数名"
"関数名" で示す割り込み関数に対するレジスタ退避・回復を制御する #pragma が不適切です。 #pragma 指定を無視します。
- C1022 (W) First operand of "演算子" is not lvalue
第 1 オペランドの"演算子"は、左辺値になりません。
- C1023 (W) Can not convert Japanese code "コード" to output type
日本語コードで指定の出力コードに変換できないものがあります。
- C1027 (W) Invalid #pragma gbr_base/gbr_base1
gbr=auto と #pragma gbr_base、 #pragma gbr_base1 が同時に指定されています。 #pragma 指定を無視します。
- C1200 (W) Division by floating point zero
定数式の中で浮動小数点数 0.0 を除数とする割り算を行っています。符号にしたがって、+ または- に対応する内部表現の値を仮定します。

12. コンパイラのエラーメッセージ

- C1201 (W) Ineffective floating point operation
定数式の中で `-`、`0.0/0.0` 等の無効演算を行っています。無効演算の結果を表わす非数に対応する内部表現の値を仮定します。
- C1300 (W) Command parameter specified twice
同じコンパイラオプションを 2 度以上指定しています。同じコンパイラオプションの中で最後に指定したものを有効とします。
- C1302 (W) "double=float" option ignored
`double=float`、`cpu=sh4` を同時に指定しています。`double=float` を無視し、`fpu=single` が指定されていると解釈してコンパイルをします。
- C1304 (W) "CPU 種別 1" is interpreted as " CPU 種別 2"
`cpu=<CPU 種別 1>`は無効です。`cpu=<CPU 種別 2>`と解釈してコンパイルします。
- C1307 (W) Section name too long
セクション名の長さが 32 文字を超えています。32 文字までを有効とし、それ以降の文字を無視します。
- C1400 (W) Function "関数名" in #pragma inline is not expanded
`#pragma inline` で指定した関数がインライン展開されませんでした。`#pragma inline` 指定を無視します。
- C1600 (W) Debugging information describing location of "名前" is lost
"名前"のシンボル情報が出力されませんでした。
- C2000 (E) Illegal preprocessor keyword
プリプロセッサ文で、誤ったキーワードを使用しています。
- C2001 (E) Illegal preprocessor syntax
プリプロセッサ文またはマクロ呼び出しの指定方法に誤りがあります。
- C2002 (E) Missing ", "
引数のある `#define` 文で引数の並びを区切るコンマ, が抜けています。
- C2003 (E) Missing ")"
名前が `#define` 文で定義されているかどうかを判定する `defined` 式で名前の次の右括弧) が抜けています。
- C2004 (E) Missing ">"
`#include` 文のファイル名の指定でファイル名の次の `>` がありません。
- C2005 (E) Cannot open include file "ファイル名"
`#include` 文で指定したファイル名のファイルがオープンできません。
- C2006 (E) Multiple #define's
`#define` 文で同じマクロ名を再定義しています。

- C2008 (E) Processor directive #elif mismatches
#elif 文に対応する#if 文、#ifdef 文、#ifndef 文、#elif 文がありません。
- C2009 (E) Processor directive #else mismatches
#else 文に対応する#if 文、#ifdef 文、#ifndef 文がありません。
- C2010 (E) Macro parameters mismatch
マクロ呼び出しの引数の数がマクロ定義の引数の数と異なります。
- C2011 (E) Line too long
マクロ展開後のソースプログラムの行が限界値を超えています。
- C2012 (E) Keyword as a macro name
プリプロセッサで規定しているキーワードを#define 文または#undef 文のマクロ名として定義しています。
- C2013 (E) Processor directive #endif mismatches
#endif 文に対応する#if 文、#ifdef 文、#ifndef 文がありません。
- C2014 (E) Missing #endif
#if 文、#ifdef 文、#ifndef 文に対応する#endif 文がないままでファイルが終了しました。
- C2016 (E) Preprocessor constant expression too complex
#if 文、#elif 文で指定した定数式の演算子と被演算子の合計が限界値を超えています。
- C2017 (E) Missing "
#include 文のファイル名の指定で、ファイル名の次に " がありません。
- C2018 (E) Illegal #line
#line 文で指定した行数が限界値を超えています。
- C2019 (E) File name too long
ファイル名の長さが限界値を超えています。
- C2020 (E) System identifier "名前" redefined
組み込み関数と同名のシンボルを定義しています。
- C2100 (E) Multiple storage classes
宣言の中で 2 つ以上の記憶クラス指定子を指定しています。
- C2101 (E) Address of register
レジスタ記憶クラスを持つ変数に対して、単項演算子&を適用しています。
- C2102 (E) Illegal type combination
型指定子の組み合わせが誤っています。

12. コンパイラのエラーメッセージ

- C2103 (E) Bad self reference structure
構造体または共用体のメンバの型を、親の構造体または共用体と同じ型で宣言しています。
- C2104 (E) Illegal bit field width
ビットフィールド幅を示す定数式が整数型ではありません。あるいはビットフィールド幅として負の整数を指定しています。
- C2105 (E) Incomplete tag used in declaration
構造体または共用体で仮宣言されたタグ名、または未宣言のタグ名を typedef 宣言、ポインタを指す型あるいは関数の返す型以外の宣言で使用しています。
- C2106 (E) Extern variable initialized
複文内で extern 記憶クラスを指定した宣言に対して初期値を指定しています。
- C2107 (E) Array of function
要素の型が関数型となる配列型を指定しています。
- C2108 (E) Function returning array
リターン値の型が配列型となる関数型を指定しています。
- C2109 (E) Illegal function declaration
複文内の関数型の変数宣言において、extern 以外の記憶クラスを指定しています。
- C2110 (E) Illegal storage class
外部定義の中で記憶クラスとして auto または register を指定しています。
- C2111 (E) Function as a member
構造体または共用体のメンバの型に関数型を指定しています。
- C2112 (E) Illegal bit field
ビットフィールドに誤った型を指定しています。ビットフィールドに許される型指定子は、char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, bool, enum のいずれか、これらの型に const または volatile を組み合わせた型です。
- C2113 (E) Bit field too wide
ビットフィールド幅が型指定子で指定したサイズ(8、16、32 ビット)を超えています。
- C2114 (E) Multiple variable declarations
変数名を同じ有効範囲の中で重複して宣言しています。
- C2115 (E) Multiple tag declarations
構造体、共用体、列挙型のタグ名を同じ有効範囲の中で重複して宣言しています。
- C2117 (E) Empty source program
ソースプログラム内に外部定義が含まれていません。

- C2118 (E) Prototype mismatch "関数名"
関数の型が以前になされている宣言で指定した型と一致しません。
- C2119 (E) Not a parameter name "引数名"
関数の引数宣言列にない識別子に対して引数宣言を行っています。
- C2120 (E) Illegal parameter storage class
関数の引数宣言で register 以外の記憶クラスを指定しています。
- C2121 (E) Illegal tag name
構造体、共用体または列挙型とタグ名の組み合わせが、以前に宣言した型とタグ名の組み合わせと異なっています。
- C2122 (E) Bit field width 0
メンバ名を指定しているビットフィールドの幅が 0 になっています。
- C2123 (E) Undefined tag name
列挙型の宣言で未定義のタグ名を使用しています。
- C2124 (E) Illegal enum value
列挙型のメンバに整数でない定数式を指定しています。
- C2125 (E) Function returning function
リターン値の型が関数型となる関数型を指定しています。
- C2126 (E) Illegal array size
配列の要素数の値が「1 以上 2147483647 以下の整数値」以外の値を指定しています。
- C2127 (E) Missing array size
配列の要素数の指定がありません。
- C2128 (E) Illegal pointer declaration for "*"
ポインタ型の宣言を示す*の直後に const、volatile 以外の型指定子を指定しています。
- C2129 (E) Illegal initializer type
変数の初期値指定において初期値の型が変数に代入可能な型ではありません。
- C2130 (E) Initializer should be constant
構造体型、共用体型、配列型の変数の初期値、または静的に割り付けられる変数の初期値に定数式でないものを指定しています。
- C2131 (E) No type nor storage class
外部データ定義において記憶クラスまたは型の指定がありません。
- C2132 (E) No parameter name
関数の引数宣言列が空であるにもかかわらず引数の宣言を行っています。

12. コンパイラのエラーメッセージ

- C2133 (E) Multiple parameter declarations
(マクロ)関数定義の引数宣言列の中で同一名の引数を重複して宣言しているか、または引数宣言が関数宣言子の中と外の 2 ヶ所で行われています。
- C2134 (E) Initializer for parameter
引数の宣言において初期値を指定しています。
- C2135 (E) Multiple initialization
同一の変数に対して、初期化を重複して行っています。
- C2136 (E) Type mismatch
extern あるいは static 記憶クラスを持つ変数あるいは関数を 2 度以上宣言しており、その型が一致していません。
- C2137 (E) Null declaration for parameter
関数の引数宣言で識別子を指定していません。
- C2138 (E) Too many initializers
構造体、共用体または配列の初期値指定において、構造体のメンバ数または配列の要素数より多く初期値の数を指定しています。あるいは、共用体の最初のメンバがスカラ型のときに 2 個以上の初期値を指定しています。
- C2139 (E) No parameter type
関数宣言の引数宣言に型指定がありません。
- C2140 (E) Illegal bit field
共用体にビットフィールドを指定しています。
- C2141 (E) Struct has no member name
構造体の先頭のメンバに無名のビットフィールドを指定しています。
- C2142 (E) Illegal void type
void 型の指定方法に誤りがあります。void 型を指定できるのは以下の三つの場合です。
(1) ポインタの指す先の型として指定する場合。
(2) 関数の返す型として指定する場合。
(3) 原型宣言の関数が引数を持たないことを明示的に指定する場合。
- C2143 (E) Illegal static function
ソースファイル内に定義のない static 記憶クラスを持つ関数宣言があります。
- C2144 (E) Type mismatch
extern 記憶クラスを持つ同じ名前の変数あるいは関数の型が一致していません。
- C2145 (E) Const/volatile specified for incomplete type
不完全型に対して const または volatile が指定されています。

- C2200 (E) Index not integer
配列の添字の式が整数型ではありません。
- C2201 (E) Cannot convert parameter "n"
関数呼び出しにおける n 番目の引数を対応する原型宣言の引数の型に変換できません。
- C2202 (E) Number of parameters mismatch
関数呼び出しにおける引数の数が原型宣言の引数の数と一致しません。
- C2203 (E) Illegal member reference for "."
演算子. の左側の式の型が構造体型または共用体型ではありません。
- C2204 (E) Illegal member reference for "->"
演算子->の左側の式の型が構造体型または共用体型へのポインタではありません。
- C2205 (E) Undefined member name
構造体、共用体への参照で宣言していないメンバ名を使用しています。
- C2206 (E) Modifiable lvalue required for "演算子"
前置または後置演算子++、--を代入可能な左辺値(配列型、const 型を除く左辺値)でない式に使用しています。
- C2207 (E) Scalar required for "!"
単項演算子! をスカラ型でない式に使用しています。
- C2208 (E) Pointer required for "*"
単項演算子* をポインタ型でない式か、または void 型へのポインタ型の式に使用しています。
- C2209 (E) Arithmetic type required for "演算子"
単項演算子+または-を算術型でない式に使用しています。
- C2210 (E) Integer required for "~"
単項演算子~ を汎整数型でない式に使用しています。
- C2211 (E) Illegal sizeof
sizeof 演算子をビットフィールドの指定のあるメンバ、関数型、void 型またはサイズの指定していない配列に使用しています。
- C2212 (E) Illegal cast
キャスト演算子で指定している型が配列型、構造体型または共用体型です。あるいはキャスト演算子の被演算子が void 型、構造体型または共用体型で型変換できません。
- C2213 (E) Arithmetic type required for "演算子"
二項演算子*、/、*=または/=を算術型でない式に適用しています。

12. コンパイラのエラーメッセージ

- C2214 (E) Integer required for "演算子"
二項演算子<<, >>, &, |, ^, %, <<=, >>=, &=, |=, ^=または%=を汎整数型でない式に適用しています。
- C2215 (E) Illegal type for "+"
二項演算子+の被演算子の型の組み合わせが許されていません。二項演算子+の型の組み合わせで許されるのは、両辺とも算術型の場合か、または一方がポインタ型で他方が汎整数型の場合だけです。
- C2216 (E) Illegal type for parameter
関数呼び出しの引数の型に void 型を指定しています。
- C2217 (E) Illegal type for "-"
二項演算子-の被演算子の型の組み合わせが許されていません。二項演算子-の型の組み合わせで許されるのは、以下の 3 つの場合です。
(1) 両方の被演算子が算術型の場合。
(2) 両方の被演算子が同じ型へのポインタ型の場合。
(3) 第 1 被演算子がポインタ型で、第 2 被演算子が汎整数型の場合。
- C2218 (E) Scalar required
条件演算子?: の第 1 被演算子の型がスカラ型ではありません。
- C2219 (E) Type not compatible in "? :"
条件演算子?: の第 2 被演算子と第 3 被演算子の型が合っていません。条件演算子?: の第 2 被演算子と第 3 被演算子の組み合わせで許されるのは、以下の 6 つの場合です。
(1) 両方とも算術型の場合。
(2) 両方とも void 型の場合。
(3) 両方とも同じ型へのポインタ型の場合。
(4) 一方がポインタ型で、他方が値 0 の整数または値 0 の整数を void 型へのポインタ型に変換したものである場合。
(5) 一方がポインタ型で、他方が void 型へのポインタ型の場合。
(6) 両方とも同じ型の構造体または共用体の場合。
- C2220 (E) Modifiable lvalue required for "演算子"
代入演算子=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=または |= の左辺の式に代入可能な左辺値(配列型、const 型を除く左辺値)以外の式を指定しています。
- C2221 (E) Illegal type for "演算子"
後置演算子++または--の被演算子にスカラ型以外の型、関数型または void 型へのポインタ型を指定しています。

- C2222 (E) Type not compatible for "="
代入演算子=の両辺の式の型が合っていません。代入演算子=の両辺の式の組み合わせで許されるのは、以下の5つの場合です。
- (1) 両方とも算術型の場合。
 - (2) 両方とも同じ型へのポインタ型の場合。
 - (3) 左辺がポインタ型で、右辺が値0の整数または値0の整数をvoid型へのポインタ型に変換したものである場合。
 - (4) 一方がポインタ型で、他方がvoid型へのポインタ型の場合。
 - (5) 両方とも同じ型の構造体または共用体の場合。
- C2223 (E) Incomplete tag used in expression
構造体または共用体で仮宣言されたタグ名を式中で使用しています。
- C2224 (E) Illegal type for assign
代入演算子+=または-=の両辺の型が正しくありません。
- C2225 (E) Undeclared name "名前"
宣言していない名前を式の中で用いています。
- C2226 (E) Scalar required for "演算子"
二項演算子&&または||をスカラ型でない式に適用しています。
- C2227 (E) Illegal type for equality
等値演算子==または!=の被演算子の型の組み合わせが許されていません。等値演算の被演算子の組み合わせで許されるのは、以下の三つの場合です。
- (1) 両方とも算術型の場合。
 - (2) 両方とも同じ型へのポインタ型の場合。
 - (3) 一方がポインタ型で、他方が値0の整数またはvoid型へのポインタ型である場合。
- C2228 (E) Illegal type for comparison
関係演算子>、<、>=または<=の被演算子の型の組み合わせが許されていません。関係演算子の被演算子の組み合わせで許されるのは、以下の二つの場合です。
- (1) 両方とも算術型の場合。
 - (2) 両方とも同じ型へのポインタ型の場合。
- C2230 (E) Illegal function call
関数呼び出しにおいて、関数型あるいは関数型へのポインタ型でない式を用いています。
- C2231 (E) Address of bit field
単項演算子&をビットフィールドに適用しています。
- C2232 (E) Illegal type for "演算子"
前置演算子++または--の被演算子にスカラ型以外の型、関数型またはvoid型へのポインタ型を指定しています。

12. コンパイラのエラーメッセージ

- C2233 (E) Illegal array reference
配列型、関数型または void 型を除くポインタ型以外の式を配列として使用しています。
- C2234 (E) Illegal typedef name reference
typedef 宣言された名前を式の中で変数として使用しています。
- C2235 (E) Illegal cast
ポインタを浮動小数点型にキャストしています。
- C2236 (E) Illegal cast in constant
定数式でポインタ型を char 型または short 型にキャストしています。
- C2237 (E) Illegal constant expression
定数式の中でポインタ型の定数を整数型へキャストした結果に対して演算を行っています。
- C2238 (E) Lvalue or function type required for "&"
単項演算子&を左辺値あるいは関数型以外の式に適用しています。
- C2239 (E) Illegal section name
セクション名に使用できない文字があります。
- C2300 (E) Case not in switch
case ラベルを switch 文以外に指定しています。
- C2301 (E) Default not in switch
default ラベルを switch 文以外に指定しています。
- C2302 (E) Multiple labels
1 つの関数内にラベル名を重複して定義しています。
- C2303 (E) Illegal continue
continue 文を while 文、for 文または do 文以外に指定しています。
- C2304 (E) Illegal break
break 文を while 文、for 文、do 文または switch 文以外に指定しています。
- C2305 (E) Void function returns value
void 型を返す関数の中の return 文でリターン値を指定しています。
- C2306 (E) Case label not constant
case ラベルの式が汎整数型の定数式ではありません。
- C2307 (E) Multiple case labels
同一の値を持つ case ラベルを 1 つの switch 文の中に重複して指定しています。
- C2308 (E) Multiple default labels
default ラベルを 1 つの switch 文の中に重複して指定しています。

- C2309 (E) No label for goto
goto 文で指定した行き先のラベルがありません。
- C2310 (E) Scalar required
while 文、for 文または do 文の制御式(文の実行を判定する式)がスカラ型ではありません。
- C2311 (E) Integer required
switch 文の制御式(文の実行を判定する式)が汎整数型ではありません。
- C2312 (E) Missing (
if 文、while 文、for 文、do 文または switch 文の制御式(文の実行を判定する式)の左括弧 "(" がありません。
- C2313 (E) Missing ;
do 文の最後のセミコロン ";" がありません。
- C2314 (E) Scalar required
if 文の制御式(文の実行を判定する式)がスカラ型ではありません。
- C2316 (E) Illegal type for return value
return 文の式の型を関数の返す型に変換することができません。
- C2400 (E) Illegal character "文字"
不正な文字があります。
- C2401 (E) Incomplete character constant
文字定数の途中で改行があります。
- C2402 (E) Incomplete string
文字列の途中で改行があります。
- C2403 (E) EOF in comment
コメントの途中でファイルが終了しました。
- C2404 (E) Illegal character code "文字コード"
不正な文字コードがあります。
- C2405 (E) Null character constant
文字定数の中に文字を指定していません。すなわち ' ' という形式の文字定数を指定しています。
- C2406 (E) Out of float
浮動小数点定数の有効桁数が 17 桁を超えています。

12. コンパイラのエラーメッセージ

- C2407 (E) Incomplete logical line
空でないソースファイルの最後の文字にバックスラッシュ"\"、またはバックスラッシュのあとに改行文字"\"[RET]"を指定しています。
- C2408 (E) Comment nest too deep
コメントのネストが 255 レベルを超えています。
- C2500 (E) Illegal token "語句"
語句の並びが文法に合っていません。
- C2501 (E) Division by zero
定数式中で整数型データのゼロ除算が行われました。
- C2600 (E) 文字列
nolistfile オプションが指定されていないならば、#error の文字列で指定されたエラーメッセージをリストファイルに表示します。
- C2650 (E) Invalid pointer reference
指定されたアドレス値が境界調整数と一致しません。
- C2700 (E) Function "関数名" in #pragma interrupt already declared
割り込み関数宣言#pragma interrupt で指定した関数が、すでに通常の関数として宣言されています。
- C2701 (E) Multiple interrupt for one function
1 つの関数に対して割り込み関数宣言#pragma interrupt を重複して宣言しています。
- C2702 (E) Multiple #pragma interrupt options
同種の割り込み仕様が重複して指定されています。
- C2703 (E) Illegal #pragma interrupt declaration
割り込み関数宣言#pragma interrupt の仕様の指定が異なります。
- C2704 (E) Illegal reference to interrupt function
割り込み関数を不正に参照しています。
- C2705 (E) Illegal parameter in interrupt function
割り込み関数で使用する引数の型が一致していません。
- C2706 (E) Missing parameter declaration in interrupt function
割り込み関数のオプション指定で使用する変数の宣言がありません。
- C2707 (E) Parameter out of range in interrupt function
割り込み関数のパラメタ tn の値が 256 を超えています。
- C2709 (E) Illegal section name declaration
#pragma section 指定に誤りがあります。

- C2710 (E) Section name too long
指定したセクション名の長さが 31 文字を超えています。
- C2711 (E) Section name table overflow
指定したセクションの数が 1 ファイルで 64 個を超えています。
- C2712 (E) GBR based displacement overflow
#pragma gbr_base、#pragma gbr_base1 で宣言した変数の領域がオーバーフローしました。
- C2713 (E) Illegal #pragma interrupt function type
#pragma interrupt 指定した関数の型が不正です。
- C2799 (E) GBR used in-line function
gbr=auto を指定した時は、GBR 関連組み込み関数は使用できません。
- C2800 (E) Illegal parameter number in in-line function
組み込み関数で使用する引数の数が一致しません。
- C2801 (E) Illegal parameter type in in-line function
組み込み関数で引数の型が一致しません。
- C2802 (E) Parameter out of range in in-line function
組み込み関数で引数の大きさが指定可能範囲を超えています。
- C2803 (E) Invalid offset value in in-line function
組み込み関数で引数の指定が不適当です。
- C2804 (E) Illegal in-line function
指定された cpu オプションでは使用できない組み込み関数があります。
- C2805 (E) Function "関数名" in #pragma inline/inline_asm already declared
"関数名" で示す関数の本体が、#pragma 指定よりも前にあります。
- C2806 (E) Multiple #pragma for one function
1 つの関数に対して複数の矛盾した #pragma 指定をしています。
- C2807 (E) Illegal #pragma inline/inline_asm declaration
#pragma inline または #pragma inline_asm の指定方法に誤りがあります。
- C2808 (E) Illegal option for #pragma inline_asm
#pragma inline_asm の指定があるにもかかわらず、code=machinecode を指定しています。
- C2809 (E) Illegal #pragma inline/inline_asm function type
#pragma inline または #pragma inline_asm を指定した識別子の種類が誤っています。

12. コンパイラのエラーメッセージ

- C2810 (E) Global variable "変数名" in #pragma gbr_base/gbr_base1 already declared
"変数名" で示す変数の定義が #pragma 指定よりも前にあります。
- C2811 (E) Multiple #pragma for one global variable
変数に対して複数の矛盾する #pragma が指定されています。
- C2812 (E) Illegal #pragma gbr_base/gbr_base1 declaration
#pragma gbr_base、#pragma gbr_base1 の指定方法が誤っています。
- C2813 (E) Illegal #pragma gbr_base/gbr_base1 global variable type
#pragma gbr_base、#pragma gbr_base1 を指定した識別子の種類が誤っています。
- C2814 (E) Function "関数名" in #pragma noregsave/noregalloc/regsave already declared
"関数名" で示す関数の本体が、#pragma 指定よりも前にあります。
- C2815 (E) Illegal #pragma noregsave/noregalloc/regsave declaration
#pragma noregsave、#pragma noregalloc、#pragma regsave の指定方法が誤っています。
- C2816 (E) Illegal #pragma noregsave/noregalloc/regsave function type
#pragma noregsave、#pragma noregalloc、#pragma regsave を指定した識別子の種類が誤っています。
- C2817 (E) Symbol "識別子" in #pragma abs16 already declared
"識別子" で示す名前の宣言が、#pragma 指定よりも前にあります。
- C2818 (E) Multiple #pragma for one symbol
同一の識別子に対して、複数の矛盾した #pragma が指定されています。
- C2819 (E) Illegal #pragma abs16 declaration
#pragma abs16 の指定方法が誤っています。
- C2820 (E) Illegal #pragma abs16 symbol type
#pragma abs16 を指定した識別子の種類が誤っています。
- C2821 (E) Global variable "変数名" in #pragma global_register already declared
#pragma global_register を指定した変数名はすでに定義されています。
- C2822 (E) Illegal register "レジスタ" in #pragma global_register
#pragma global_register を指定したレジスタが不正です。
- C2823 (E) Illegal #pragma global_register declaration
#pragma global_register の指定方法が誤っています。

- C2824 (E) Illegal #pragma global_register type
#pragma global_register を指定できない変数が指定されています。
- C2828 (E) Illegal #pragma entry declaration
#pragma entry 宣言に構文エラーがあります。
- C2829 (E) Function "関数名" in #pragma entry already declared
#pragma entry 宣言の前に同名のシンボルがあるか、または既に#pragma 指定されています。
- C2830 (E) Illegal #pragma entry function type
指定されたシンボルが関数ではありません。
- C2831 (E) Multiple #pragma entry declaration
#pragma entry 宣言が 3 つ以上存在しています。
- C2832 (E) Illegal #pragma stacksize declaration
#pragma stacksize 宣言に構文エラーがあります。
- C2833 (E) Multiple #pragma stacksize declaration
#pragma stacksize 宣言が複数存在しています。
- C3000 (F) Statement nest too deep
if 文、while 文、for 文、do 文および switch 文のネストが、限界値を超えています。
- C3006 (F) Too many parameters
関数の宣言または呼び出しにおいて引数の数が限界値を超えています。
- C3007 (F) Too many macro parameters
マクロの定義または呼び出しにおいて、引数の数が限界値を超えています。
- C3008 (F) Line too long
マクロ展開後の 1 行の長さが限界値を超えています。
- C3009 (F) String literal too long
文字列の長さが 32766 文字を超えています。文字列の長さは、連続して指定した文字列を連結した後のバイト数です。ここでいう文字列の長さとは、ソースプログラム上の長さではなく文字列のデータに含まれるバイト数で、拡張表記も 1 文字に数えます。
- C3013 (F) Too many switches
switch 文の数が限界値を超えています。
- C3014 (F) For nest too deep
for 文のネストが限界値を超えています。
- C3015 (F) Symbol table overflow
コンパイラが生成するシンボルの数が限界値を超えています。

12. コンパイラのエラーメッセージ

- C3016 (F) Internal label overflow
コンパイラが生成する内部ラベルの数が限界値を超えています。
- C3017 (F) Too many case labels
1 つの switch 文の中の case ラベルの数が限界値を超えています。
- C3018 (F) Too many goto labels
1 つの関数の中で定義している goto ラベルの数が限界値を超えています。
- C3019 (F) Cannot open source file "ファイル名"
ソースファイルをオープンすることができません。
- C3020 (F) Source file input error "ファイル名"
ソースファイルまたはインクルードファイルを読み込むことができません。
- C3021 (F) Memory overflow
コンパイラが内部で使用するメモリ領域を割り当てることができません。
- C3022 (F) Switch nest too deep
switch 文のネストが限界値を超えています。
- C3023 (F) Type nest too deep
基本型を修飾する型 (ポインタ型、配列型、関数型) の数が 16 個を超えています。
- C3024 (F) Array dimension too deep
配列の次元数が 6 次元を超えています。
- C3025 (F) Source file not found
コマンドラインの中にソースファイル名の指定がありません。
- C3026 (F) Expression too complex
式が複雑すぎます。
- C3027 (F) Source file too complex
プログラムの文のネストが深いかあるいは、式が複雑すぎます。
- C3031 (F) Data size overflow
配列または構造体の大きさが、2147483647 バイトを超えています。
- C3100 (F) Misaligned pointer access
境界整合が正しくないポインタを用いて参照または設定をしようとしています。
- C3201 (F) Object size overflow
オブジェクトサイズが 4G バイトを超えています。
- C3203 (F) Assembly source line too long
出力するアセンブリソースの 1 行が長すぎます。

C3204 (F) Illegal stack access

関数内で使用するスタックのサイズ(局所変数領域、レジスタ退避領域その他関数呼び出しのためのパラメタプッシュ領域等含む)または、その関数呼び出しのためのパラメタ領域が 2G バイトを超えています。

C3300 (F) Cannot open internal file

以下、3つの場合のいずれかでエラーが起こっている可能性があります。

- (1) 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。
- (2) 中間ファイルと同じ名前のファイルが既に存在しています。
- (3) コンパイラが内部で使用するファイルをオープンすることができません。

C3301 (F) Cannot close internal file

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。コンパイラのインストール手順に誤りがないことを確認してください。

C3302 (F) Cannot input internal file

コンパイラが内部で生成する中間ファイルを読み込むことができません。コンパイラのインストール手順に誤りがないことを確認してください。

C3303 (F) Cannot output internal file

コンパイラが内部で生成する中間ファイルに書き込むことができません。ディスクの空き容量を増やしてください。

C3304 (F) Cannot delete internal file

コンパイラが内部で生成する中間ファイルを削除することができません。コンパイラが生成する中間ファイルをアクセスしていないかを確認してください。

C3305 (F) Invalid command parameter "オプション"

コンパイラオプションの指定方法が誤っています。

C3306 (F) Interrupt in compilation

コンパイル処理中に標準入力端末から (CNTL)+C コマンドによる割り込みを検出しました。

C3307 (F) Compiler version mismatch

コンパイラを構成するファイル間のバージョンが一致していません。インストールガイドの組み込み方法を参照し、コンパイラ本体を再インストールしてください。

C3308 (F) Cannot create file "ファイル名"

コンパイラが生成するファイルを作成できません。

C3320 (F) Command parameter buffer overflow

コマンドラインの指定が 4096 文字を超えています。

12. コンパイラのエラーメッセージ

C3321 (F) Illegal environment variable

以下の 4 つの場合のいずれかでエラーが起っています。

- (1) 環境変数 SHC_LIB が設定されていません。
- (2) 環境変数 SHC_LIB にコンパイラの実行ファイルパス名が指定されていません。
- (3) 環境変数 SHC_LIB の設定でファイル名の規約に反した指定をしているか、パス名の長さが 118 文字を超えています。
- (4) 環境変数 SHCPU に、"SH1", "SH2", "SH2E", "SHDSP", "SH3", "SH3DSP", "SH4" 以外の設定がされています。

C4000-C4999 (-) Internal error

コンパイラの内部処理で何らかの障害が生じました。本コンパイラをお求めになった営業所あるいは代理店にエラーの発生状況をご連絡ください。

C5003 (F) #include file "ファイル名" includes itself

自分自身のファイル"ファイル名"をインクルードしています。

C5004 (F) Out of memory

コンパイルに必要なメモリが不足しています。システムのメモリを増やすか、他のアプリケーションを終了してください。

C5005 (F) Could not open source file "名前"

ファイル"名前"をオープンできませんでした。ファイル名が正しいか確認してください。

C5006 (E) Comment unclosed at end of file

コメントの終了指定*/がありません。

C5007 (E) (I) Unrecognized token

認識できない字句があります(マクロの場合は(I)となります)。

C5008 (E) (I) Missing closing quote

文字列の終了指定 " がありません(マクロの場合は(I)となります)。

C5009 (I) Nested comment is not allowed

/* */コメントがネストしています。

C5010 (E) "#" not expected here

#が行の先頭、プリプロセッサ以外に指定されています。

C5011 (E) Unrecognized preprocessing directive

認識できないプリプロセッサのキーワードがあります。

C5012 (E) Parsing restarts here after previous syntax error

字句の解析を再開しました。

C5013 (E) (F) Expected a file name

ファイル名が必要です。

#include 文では(F)、#line 文では(E)となります。

- C5014 (E) Extra text after expected end of preprocessing directive
プリプロセッサ文の後にさらにテキストが記述されています。
- C5016 (F) "名前" is not a valid source file name
ファイル"名前"が有効ではありません。
- C5017 (E) Expected a "]"
"]"がありません。
- C5018 (E) Expected a ")"
")"がありません。
- C5019 (E) Extra text after expected end of number
数値の後ろにさらにテキストが記述されています。
- C5020 (E) Identifier "名前" is undefined
シンボル"名前"の定義がありません。
- C5021 (W) Type qualifiers are meaningless in this declaration
意味のない型限定子を指定しています。型限定子を無効にします。
- C5022 (E) Invalid hexadecimal number
16進数の記述に誤りがあります。
- C5024 (E) Invalid octal digit
8進数の記述に誤りがあります。
- C5025 (E) Quoted string should contain at least one character
文字定数が空です。
- C5026 (E) Too many characters in character constant
文字定数中の文字数が多すぎます。
- C5027 (W) Character value is out of range
文字の値が範囲を超えています。超えた値は切り捨てられます。
- C5028 (E) Expression must have a constant value
式の値が定数ではありません。
- C5029 (E) Expected an expression
式が必要です。
- C5030 (E) Floating constant is out of range
浮動小数点数の値が範囲を超えています。
- C5031 (E) Expression must have integral type
式の型は整数型でなければなりません。

12. コンパイラのエラーメッセージ

- C5032 (E) Expression must have arithmetic type
式の型は算術型でなければなりません。
- C5033 (E) Expected a line number
#line 文には行番号が必要です。
- C5034 (E) Invalid line number
#line 文の行番号が有効ではありません。
- C5035 (F) #error directive: "行番号"
#error 文が適用されました。
- C5036 (E) The #if for this directive is missing
#if 文の指定方法に誤りがあります。
- C5037 (E) The #endif for this directive is missing
#endif 文の指定方法に誤りがあります。
- C5038 (W) Directive is not allowed -- an #else has already appeared
#else 文はすでに出現しました。本指定を読み飛ばします。
- C5039 (E) Division by zero
ゼロ除算が発生しました。
- C5040 (E) Expected an identifier
識別子が必要です。
- C5041 (E) Expression must have arithmetic or pointer type
式の型は算術型またはポインタ型でなければなりません。
- C5042 (E) Operand types are incompatible ("型 1" and "型 2")
"型 1"と"型 2"のオペランドの型が適合しません。
- C5044 (E) Expression must have pointer type
式の型はポインタ型でなければなりません。
- C5045 (W) #undef may not be used on this predefined name
システムで定義しているマクロ名を取り消すことはできません。#undef 指定を無効にします。
- C5046 (W) This predefined name may not be redefined
システムで定義しているマクロ名を再定義することはできません。#define 指定を無効にします。
- C5047 (W) Incompatible redefinition of macro "名前" (declared at line "行番号")
マクロ"名前"の再定義が以前の定義の形式と異なります。本マクロ定義を無効にします。

- C5049 (E) Duplicate macro parameter name
マクロのパラメタ名を 2 重定義しています。
- C5050 (E) "##" may not be first in a macro definition
#define マクロの最初に##が指定されています。
- C5051 (E) "##" may not be last in a macro definition
#define マクロの最後に##が指定されています。
- C5052 (E) Expected a macro parameter name
#に続くマクロ引数がありません。
- C5053 (E) Expected a ":"
":"が必要です。
- C5054 (W) Too few arguments in macro invocation
マクロ展開時の実引数が足りません。
- C5055 (W) Too many arguments in macro invocation
マクロ展開時の実引数が多すぎます。
- C5056 (E) Operand of sizeof may not be a function
sizeof 演算のオペランドに関数を指定できません。
- C5057 (E) This operator is not allowed in a constant expression
この演算子は定数式中に指定できません。
- C5058 (E) This operator is not allowed in a preprocessing expression
この演算子はプリプロセッサの式中で指定できません。
- C5059 (E) Function call is not allowed in a constant expression
定数式中で関数呼び出しはできません。
- C5060 (E) This operator is not allowed in an integral constant expression
この演算子は整数型定数式中で指定できません。
- C5061 (W) Integer operation result is out of range
整数演算の結果が値の範囲を超えました。オーバフローした上位ビットを無視した値を仮定します。
- C5062 (W) Shift count is negative
シフトカウントが負の値です。指定された通りに演算します。
- C5063 (W) Shift count is too large
シフトカウントが有効ビット数を超えています。指定された通りに演算します。

12. コンパイラのエラーメッセージ

- C5064 (W) Declaration does not declare anything
宣言を指定するシンボルがありません。宣言を無視します。
- C5065 (E) Expected a ";"
";"が必要です。
- C5066 (E) Enumeration value is out of "int" range
enum 型メンバの値が int 型の範囲を超えました。
- C5067 (E) Expected a "}"
"}"が必要です。
- C5068 (W) Integer conversion resulted in a change of sign
符号変換を伴った整数型変換が実施されました。ビット列をそのまま設定します。
- C5069 (W) Integer conversion resulted in truncation
上位バイト側を切り捨てる整数型変換が実施されました。切り捨て後の値を設定します。
- C5070 (E) Incomplete type is not allowed
不完全型が指定されています。
- C5071 (E) Operand of sizeof may not be a bit field
sizeof 演算子のオペランドにビットフィールドが指定されています。
- C5075 (E) Operand of "*" must be a pointer
*演算子のオペランドの型がポインタ型ではありません。
- C5077 (E) This declaration has no storage class or type specifier
記憶クラスまたは型の指定がありません。
- C5079 (E) Expected a type specifier
型指定子が必要です。
- C5080 (E) A storage class may not be specified here
ここでは記憶クラスを指定することはできません。
- C5081 (E) More than one storage class may not be specified
記憶クラスを複数指定することはできません。
- C5083 (W) Type qualifier specified more than once
const/volatile 限定子を複数指定しています。余分な指定を無視します。
- C5084 (E) Invalid combination of type specifiers
型の組み合わせが正しくありません。
- C5085 (E) Invalid storage class for a parameter
仮引数に不当な記憶クラスを指定しています。

- C5086 (E) Invalid storage class for a function
関数に不当な記憶クラスを指定しています。
- C5087 (E) A type specifier may not be used here
型を指定することはできません。
- C5088 (E) Array of functions is not allowed
関数を要素とする配列は指定できません。
- C5089 (E) Array of void is not allowed
void 型を要素とする配列は指定できません。
- C5090 (E) Function returning function is not allowed
関数型をリターン型とする関数は指定できません。
- C5091 (E) Function returning array is not allowed
配列をリターン型とする関数は指定できません。
- C5093 (E) Function type may not come from a typedef
typedef 宣言された関数型を使用することはできません。
- C5094 (E) The size of an array must be greater than zero
配列のサイズは 0 より大きな値でなければなりません。
- C5095 (E) Array is too large
配列のサイズが大きすぎます。
- C5097 (E) A function may not return a value of this type
関数はこの型の値を返すことができません。
- C5098 (E) An array may not have elements of this type
配列はこの型を要素とすることができません。
- C5100 (E) Duplicate parameter name
仮引数の名前が重複しています。
- C5101 (E) "名前" has already been declared in the current scope
同スコープ内にすでに"名前"の宣言が存在します。
- C5103 (E) Class is too large
クラスのサイズが大きすぎます。
- C5105 (E) Invalid size for bit field
ビットフィールドのサイズが不正です。
- C5106 (E) Invalid type for a bit field
ビットフィールドの型が不正です。

12. コンパイラのエラーメッセージ

- C5107 (E) Zero-length bit field must be unnamed
長さ 0 のビットフィールドには名前をつけられません。
- C5108 (W) Signed bit field of length 1
符号付整数型の長さ 1 のビットフィールドが指定されています。指定された型で処理します。
- C5109 (E) Expression must have (pointer-to-) function type
式は関数型へのポインタ型でなければなりません。
- C5110 (E) Expected either a definition or a tag name
宣言の定義またはタグ名が必要です。
- C5111 (I) Statement is unreachable
実行されない文です。最適化により削除される可能性があります。
- C5112 (E) Expected "while"
while キーワードが必要です。
- C5114 (E) Entity-kind "名前" was referenced but not defined
参照される"名前"の定義がありません。
- C5115 (E) A continue statement may only be used within a loop
continue 文はループの中で有効です。
- C5116 (E) A break statement may only be used within a loop or switch
break 文はループまたは switch 文の中で有効です。
- C5117 (W) Non-void entity-kind "名前" should return a value
void 型でない関数がリターン値を返しません。リターン値は不定です。
- C5118 (E) A void function may not return a value
void 型を返す関数はリターン値を返すことはできません。
- C5119 (E) Cast to type "型" is not allowed
"型"へのキャストは指定できません。
- C5120 (E) Return value type does not match the function type
リターン値と関数の型が合いません。
- C5121 (E) A case label may only be used within a switch
case ラベルを switch 文以外で使用しています。
- C5122 (E) A default label may only be used within a switch
default ラベルを switch 文以外で使用しています。

- C5123 (E) Case label value has already appeared in this switch case
ラベルの値がすでに switch 文の中に存在します。
- C5124 (E) Default label has already appeared in this switch default
ラベルの値がすでに switch 文の中に存在します。
- C5125 (E) Expected a "("
"("が必要です。
- C5126 (E) Expression must be an lvalue
式は左辺値でなければなりません。
- C5127 (E) Expected a statement
文が必要です。
- C5128 (I) Loop is not reachable from preceding code
実行されないループ文です。
- C5129 (E) A block-scope function may only have extern storage class
ブロック内で宣言された関数は extern 記憶クラスでなければなりません。
- C5130 (E) Expected a "{"
"{"が必要です。
- C5131 (E) Expression must have pointer-to-class type
式はクラスへのポインタ型でなければなりません。
- C5132 (E) Expression must have pointer-to-struct-or-union type
式は構造体または共用体へのポインタ型でなければなりません。
- C5133 (E) Expected a member name
メンバ名が必要です。
- C5134 (E) Expected a field name
フィールド名が必要です。
- C5135 (E) Entity-kind "名前" has no member "メンバ名"
"名前"は"メンバ名"を持ちません。
- C5136 (E) Entity-kind "名前" has no field "フィールド名"
"名前"は"フィールド名"を持ちません。
- C5137 (E) Expression must be a modifiable lvalue
式は修正可能な左辺値でなければなりません。
- C5139 (E) Taking the address of a bit field is not allowed
ビットフィールドのアドレスを参照することはできません。

12. コンパイラのエラーメッセージ

- C5140 (E) Too many arguments in function call
関数呼び出しの実引数の数が多すぎます。
- C5142 (E) Expression must have pointer-to-object type
式はオブジェクトへのポインタ型でなければなりません。
- C5143 (F) Program too large or complicated to compile
プログラムが大きすぎるかまたは複雑すぎます。
- C5144 (E) A value of type "型 1" cannot be used to initialize an entity of type "型 2"
初期値の"型 1"と変数の"型 2"が異なります。
- C5145 (E) Entity-kind "名前" may not be initialized
"名前"を初期化することはできません。
- C5146 (E) Too many initializer values
初期値の数が多すぎます。
- C5147 (E) Declaration is incompatible with "名前" (declared at line "行番号")
前に宣言した"名前"の型が合致しません。
- C5148 (E) Entity-kind "名前" has already been initialized
すでに"名前"の初期値が設定されています。
- C5149 (E) A global-scope declaration may not have this storage class
大域的なスコープでの宣言にはこの記憶クラスを指定できません。
- C5150 (E) A type name may not be redeclared as a parameter
型名を仮引数で再宣言することはできません。
- C5151 (E) A typedef name may not be redeclared as a parameter
型名を仮引数で再宣言することはできません。
- C5153 (E) Expression must have class type
式はクラス型でなければなりません。
- C5154 (E) Expression must have struct or union type
式は構造体または共用体型でなければなりません。
- C5157 (E) Expression must be an integral constant expression
式は整数型の定数式でなければなりません。
- C5158 (E) Expression must be an lvalue or a function designator
式は左辺値または関数名でなければなりません。

- C5159 (E) Declaration is incompatible with previous "名前" (declared at line "行番号")
前に使用した"名前"の型と合致しません。
- C5160 (E) Name conflicts with previously used external name "名前"
前に使用した外部名"名前"と名前が重なります。
- C5161 (I) Unrecognized #pragma
認識できない#pragma 指定があります。#pragma 指定を無視します。
- C5163 (F) Could not open temporary file "名前"
テンポラリファイル"名前"をオープンできませんでした。コンパイラ的环境設定やホスト環境のファイルシステム異常がないか確認ください。
- C5164 (F) Name of directory for temporary files is too long ("名前")
テンポラリファイルの"名前"が長すぎます。
- C5165 (E) Too few arguments in function call
関数呼び出しの実引数の数が足りません。
- C5166 (E) Invalid floating constant
浮動小数点定数の指定が不正です。
- C5167 (E) Argument of type "型 1" is incompatible with parameter of type "型 2"
実引数の型"型 1"と仮引数の型"型 2"とが合致しません。
- C5168 (E) A function type is not allowed here
関数型は許されません。
- C5169 (E) Expected a declaration
宣言が必要です。
- C5170 (W) Pointer points outside of underlying object
ポインタが指している領域がオブジェクトの範囲を超えています。
- C5171 (E) Invalid type conversion
キャストの型が不正です。
- C5172 (I) External/internal linkage conflict with previous declaration
前の宣言と外部/内部リンケージが異なります。内部リンケージが仮定されます
- C5173 (E) Floating-point value does not fit in required integral type
浮動小数点数型の値を整数型に変換するときに値の範囲を超えました。
- C5174 (I) Expression has no effect
効果のない式です。最適化で削除される可能性があります。

12. コンパイラのエラーメッセージ

- C5175 (W) Subscript out of range
配列のインデックスが範囲を超えています。指定されたインデックスで処理を継続します。
- C5177 (W) Entity-kind "entity" was declared but never referenced
参照されない宣言があります。
- C5179 (W) Right operand of "%" is zero
%演算子の右边が値 0 です。指定された式で評価します。
- C5182 (F) Could not open source file "名前" (no directories in search list)
ファイル"名前"をオープンできませんでした。ディレクトリが存在するかどうか確認してください。
- C5183 (E) Type of cast must be integral
キャストの型は整数型でなければなりません。
- C5184 (E) Type of cast must be arithmetic or pointer
キャストの型は算術型またはポインタ型でなければなりません。
- C5185 (I) Dynamic initialization in unreachable code
初期化式は実行されません。実行時に初期値は設定されません。
- C5186 (W) Pointless comparison of unsigned integer with zero
0 と符号無し整数の無意味な比較をしています。指定された通りに式を評価します。
- C5187 (I) Use of "=" where "==" may have been intended
"=="が意図される式で"="が使われています。指定された通りに式を評価します。
- C5189 (F) Error while writing "ファイル名" file
ファイルの書き込みに失敗しました。
- C5191 (W) Type qualifier is meaningless on cast type
キャストの型に意味のない型限定子を指定しています。指定された型を無視します。
- C5192 (W) Unrecognized character escape sequence
認識できないエスケープシーケンス文字を指定しています。値をそのまま使用します。
- C5193 (I) Zero used for undefined preprocessing identifier
プリプロセッサ文の式評価に値 0 が使われました。指定された通りに式を評価します。
- C5219 (F) Error while deleting file "ファイル名"
ファイル"ファイル名"を削除することができません。
- C5221 (W) Floating-point value does not fit in required floating-point type
浮動小数点型を要求された浮動小数点型に変換できません。無限大の値とみなします。

- C5224 (W) The format string requires additional arguments
フォーマット文字列で要求する引数より実引数の数が足りません。
- C5225 (W) The format string ends before this argument
フォーマット文字列が要求する引数より実引数の数が多すぎます。
- C5226 (W) Invalid format string conversion
フォーマット変換の形式が実引数の型と異なります。
- C5229 (W) Bit field cannot contain all values of the enumerated type
ビットフィールドが enum 型全ての値を保持できません。結果は切り捨てられます。
- C5235 (E) Variable "名前" was declared with a never-completed type
変数"名前"が不完全型のまま宣言されました。
- C5236 (W) (I) Controlling expression is constant
制御式が定数です(I)。制御式がアドレス定数です(W)。指定された通りに式を評価します。
- C5237 (I) Selector expression is constant
switch 文の制御式が定数です。
- C5238 (E) Invalid specifier on a parameter
引数宣言で不正な指定子を使用しています。
- C5239 (E) Invalid specifier outside a class declaration
クラス宣言外で不正な指定子を使用しています。
- C5240 (E) Duplicate specifier in declaration
1 つの宣言内で指定子を重複して使用しています。
- C5241 (E) A union is not allowed to have a base class
union 型は基底クラスを持つことはできません。
- C5242 (E) Multiple access control specifiers are not allowed
アクセス指定子が重複して使われています。
- C5243 (E) Class or struct definition is missing
class 定義の括弧の対応がとれません。
- C5244 (E) Qualified name is not a member of class "型" or its base classes
限定名がクラスまたは基底クラスのメンバの"型"ではありません。
- C5245 (E) A nonstatic member reference must be relative to a specific object
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。

12. コンパイラのエラーメッセージ

- C5246 (E) A nonstatic data member may not be defined outside its class
非静的データメンバはクラス外で定義できません。
- C5247 (E) Entity-kind "名前" has already been defined
"名前"はすでに定義されています。
- C5248 (E) Pointer to reference is not allowed
リファレンス型へのポインタ型は許されません
- C5249 (E) Reference to reference is not allowed
リファレンス型へのリファレンス型は許されません。
- C5250 (E) Reference to void is not allowed
void 型へのリファレンス型は許されません。
- C5251 (E) Array of reference is not allowed
リファレンス型の配列は許されません。
- C5252 (E) Reference entity-kind "名前" requires an initializer
リファレンス型の定義"名前"には初期値が必要です。
- C5253 (E) Expected a ", "
カンマ", "が必要です。
- C5254 (E) Type name is not allowed
型名は許されません。
- C5255 (E) Type definition is not allowed
型の定義は許されません。
- C5256 (E) Invalid redeclaration of type name "名前" (declared at line
"行番号")
型名"名前"を再定義することはできません。
- C5257 (E) Const entity-kind "名前" requires an initializer
const 型の定義"名前"には初期値が必要です。
- C5258 (E) "this" may only be used inside a nonstatic member function
"this"が非静的メンバ関数以外で使われています。
- C5259 (E) Constant value is not known
const 型の値が不明です。
- C5261 (I) Access control not specified ("名前" by default)
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。

- C5262 (E) Not a class or struct name
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。
- C5263 (E) Duplicate base class name
基底クラスを二重に指定しています。
- C5264 (E) Invalid base class
基底クラスが不正です。
- C5265 (E) Entity-kind "名前" is inaccessible
"名前"をアクセスすることはできません。
- C5266 (E) "名前" is ambiguous
指定された"名前"があいまいです。
- C5269 (E) Implicit conversion to inaccessible base class "型" is not allowed
アクセス不可能なクラスへの暗黙の型変換は許されません。
- C5274 (E) Improperly terminated macro invocation
マクロ呼び出しの途中でファイルが終了しました。
- C5276 (E) Name followed by "::" must be a class or namespace name
::に続く名前はクラス名または namespace 名でなければなりません。
- C5277 (E) Invalid friend declaration
フレンド宣言の指定が正しくありません。
- C5278 (E) A constructor or destructor may not return a value
コンストラクタやデストラクタはリターン値を持ってません。
- C5279 (E) Invalid destructor declaration
デストラクタの宣言が正しくありません。
- C5280 (E)(W) Declaration of a member with the same name as its class
クラス名と同じ名前のメンバ名を宣言しています。
(W) 非 static 変数名
(E) static 変数名, typedef 名, enum メンバなど
- C5281 (E) Global-scope qualifier (leading "::") is not allowed
グローバルなスコープ決定演算子は許されません。
- C5282 (E) The global scope has no "名前"
"名前"がグローバルなスコープに宣言されていません。
- C5283 (E) Qualified name is not allowed
限定名は許されません。

12. コンパイラのエラーメッセージ

- C5284 (W) NULL reference is not allowed
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。
- C5285 (E) Initialization with "{...}" is not allowed for object of type
"型"
"型"のオブジェクトに{}形式の初期化は許されません。
- C5286 (E) Base class "type" is ambiguous
基底クラスの型があいまいです。
- C5287 (E) Derived class "type" contains more than one instance of class
"型"
派生型が複数の同一クラス"型"を含みます。
- C5288 (E) Cannot convert pointer to base class "型 1" to pointer to derived
class "型 2" -- base class is virtual
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。
- C5289 (E) No instance of constructor "名前" matches the argument list
コンストラクタ"名前"の引数が一致しません。
- C5290 (E) Copy constructor for class "型" is ambiguous
クラス"型"のコピーコンストラクタがあいまいです。
- C5291 (E) No default constructor exists for class "型"
クラス"型"のデフォルトコンストラクタは存在しません。
- C5292 (E) "名前" is not a nonstatic data member or base class of class
"型"
"名前"が非静的データメンバまたは基底クラス"型"ではありません。
- C5293 (E) Indirect nonvirtual base class is not allowed
仮想でない間接基底クラスは許されません。
- C5294 (E) Invalid union member -- class "型" has a disallowed member function
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。
- C5297 (E) Expected an operator
演算子が必要です。
- C5298 (E) Inherited member is not allowed
継承されたメンバを使用することはできません。
- C5299 (E) Cannot determine which instance of entity-kind "名前" is intended
オーバーロード関数の"名前"を決定できません。

- C5300 (E) A pointer to a bound function may only be used to call the function
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。
- C5302 (E) Entity-kind "名前" has already been defined
関数"名前"はすでに定義されています。
- C5304 (E) No instance of entity-kind "名前" matches the argument list
関数"名前"の引数が一致しません。
- C5305 (E) Type definition is not allowed in function return type declaration
関数のリターン型の宣言で型の定義をすることはできません。
- C5306 (E) Default argument not at end of parameter list
デフォルト引数の宣言がパラメタリストの最後ではありません。
- C5307 (E) Redefinition of default argument
デフォルト引数を再定義しています。
- C5308 (E) More than one instance of entity-kind "名前" matches the argument
list:
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。
- C5309 (E) More than one instance of constructor "名前" matches the argument
list:
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。
- C5310 (E) Default argument of type "型 1" is incompatible with parameter
of type "型 2"
デフォルト値の"型 1"が引数の"型 2"に合致しません。
- C5311 (E) Cannot overload functions distinguished by return type alone
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。
- C5312 (E) No suitable user-defined conversion from "型 1" to "型 2" exists
適切な利用者定義変換"型 1"から"型 2"が存在しません。
- C5313 (E) Type qualifier is not allowed on this function
関数に型限定子(const,volatile)を指定することはできません。
- C5314 (E) Only nonstatic member functions may be virtual
静的メンバ関数にvirtualを指定しています。
- C5315 (E) The object has type qualifiers that are not compatible with the
member function
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。

12. コンパイラのエラーメッセージ

C5316 (E) Program too large to compile (too many virtual functions)
仮想関数の数が多すぎます。

C5317 (E) Return type is not identical to nor covariant with return type
"型" of overridden virtual function entity-kind "名前"
仮想関数"名前"のリターン型"型"が異なります。

C5318 (E) Override of virtual entity-kind "名前" is ambiguous
仮想関数"名前"の置き換えがあいまいです。

C5319 (E) Pure specifier ("= 0") allowed only on virtual functions
純粋指定子"=0"を仮想関数以外に指定しています。

C5320 (E) Badly-formed pure specifier (only "= 0" is allowed)
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。

C5321 (E) Data member initializer is not allowed
データメンバの初期化指定が正しくありません。

C5322 (E) Object of abstract class type "型" is not allowed:
抽象クラス"型"のオブジェクトは定義できません。

C5323 (E) Function returning abstract class "型" is not allowed:
抽象クラス"型"を返す関数は定義できません。

C5324 (I) Duplicate friend declaration
フレンド宣言が重複して指定されています。

C5325 (E) Inline specifier allowed on function declarations only
inline 指定子は関数宣言でのみ有効です。

C5326 (E) "inline" is not allowed
inline 指定は許されません。

C5327 (E) Invalid storage class for an inline function
inline 関数の記憶クラスが不正です。

C5328 (E) Invalid storage class for a class member
クラスメンバの記憶クラスが不正です。

C5329 (E) Local class member entity-kind "名前" requires a definition
局所クラスメンバ"名前"の定義がありません。

C5330 (E) Entity-kind "名前" is inaccessible
"名前"をアクセスできません。

- C5332 (E) Class "type" has no copy constructor to copy a const object
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。
- C5333 (E) Defining an implicitly declared member function is not allowed
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。
- C5334 (E) Class "型" has no suitable copy constructor
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。
- C5335 (E) Linkage specification is not allowed
リンケージ指定子を指定することはできません。
- C5336 (E) Unknown external linkage specification
認識できないリンケージ指定が指定されました。
- C5337 (E) Linkage specification is incompatible with previous "名前"
(declared at line "行番号")
前に指定されたリンケージ指定子"名前"と合致しません。
- C5338 (E) More than one instance of overloaded function "名前" has "C" linkage
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。
- C5339 (E) Class "型" has more than one default constructor
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。
- C5341 (E) "operator 演算子" must be a member function
演算子関数"演算子"はメンバ関数でなければなりません。
- C5342 (E) Operator may not be a static member function
静的メンバ関数の演算子関数は許されません。
- C5343 (E) No arguments allowed on user-defined conversion
利用者定義変換に引数は許されません。
- C5344 (E) Too many parameters for this operator function
演算子関数の引数の数が多すぎます。
- C5345 (E) Too few parameters for this operator function
演算子関数の引数の数が足りません。
- C5346 (E) Nonmember operator requires a parameter with class type
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。
- C5347 (E) Default argument is not allowed
デフォルト引数は許されません。

12. コンパイラのエラーメッセージ

- C5348 (E) More than one user-defined conversion from "型 1" to "型 2" applies:
"型 1"から"型 2"への利用者定義型変換があいまいです。
- C5349 (E) No operator "演算子" matches these operands
演算子関数"演算子"のオペランドが一致しません。
- C5350 (E) More than one operator "演算子" matches these operands:
演算子関数"演算子"のオペランドがあいまいです。
- C5351 (E) First parameter of allocation function must be of type "size_t"
operator new の第 1 パラメタは size_t 型でなければなりません。
- C5352 (E) Allocation function requires "void *" return type
operator new のリターン型は void *型でなければなりません。
- C5353 (E) Deallocation function requires "void" return type
operator delete のリターン型は void 型でなければなりません。
- C5354 (E) First parameter of deallocation function must be of type
"void *"
operator delete の第 1 パラメタは void *型でなければなりません。
- C5356 (E) Type must be an object type
型はオブジェクト型でなければなりません。
- C5357 (E) Base class "type" has already been initialized
基底クラスはすでに初期化されています。
- C5359 (E) Entity-kind "名前" has already been initialized
"名前"はすでに初期化されています。
- C5360 (E) Name of member or base class is missing
メンバ名または基底クラスに誤りがあります。
- C5363 (E) Invalid anonymous union -- nonpublic member is not allowed
無名 union のメンバが公開メンバではありません。
- C5364 (E) Invalid anonymous union -- member function is not allowed
無名 union にメンバ関数は許されません。
- C5365 (E) Anonymous union at global or namespace scope must be declared
static
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。
- C5366 (E) Entity-kind "名前" provides no initializer for:
"名前"に初期化指定はできません。

- C5367 (E) Implicitly generated constructor for class "型" cannot initialize:
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。
- C5368 (W) Entity-kind "名前" defines no constructor to initialize the following:
"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。
- C5369 (E) Entity-kind "名前" has an uninitialized const or reference member
"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。
- C5370 (W) Entity-kind "名前" has an uninitialized const field
"名前"の const フィールドが初期化されていません。
- C5371 (E) Class "型" has no assignment operator to copy a const object
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。
- C5372 (E) Class "型" has no suitable assignment operator
クラス"型"に適切な代入演算が定義されていません。
- C5373 (E) Ambiguous assignment operator for class "型"
クラス"型"の代入演算子関数があいまいです。
- C5375 (E) Declaration requires a typedef name
typedef 名の宣言が必要です。
- C5377 (E) "virtual" is not allowed
virtual を指定することはできません。
- C5378 (E) "static" is not allowed
static を指定することはできません。
- C5380 (E) Expression must have pointer-to-member type
式はメンバへのポインタ型でなければなりません。
- C5381 (I) Extra ";" ignored
余分な";"を無視します。
- C5382 (W) Nonstandard member constant declaration (standard form is a static
const integral member)
const メンバの宣言が標準形式ではありません。初期化は無効です。
- C5384 (E) No instance of overloaded "名前" matches the argument list
オーバーロード関数"名前"の引数リストが一致しません。
- C5386 (E) No instance of entity-kind "名前" matches the required type
要求される型のオーバーロード関数"名前"がありません。

12. コンパイラのエラーメッセージ

- C5388 (E) "operator->" for class "型 1" returns invalid type "型 2"
クラス"型 1"の operator->演算関数のリターン型"型 2"が正しくありません。
- C5389 (E) A cast to abstract class "型" is not allowed:
抽象クラス"型"へのキャストは許されません。
- C5391 (E) A new-initializer may not be specified for an array
配列を new によって初期化することはできません。
- C5392 (E) Member function "名前" may not be redeclared outside its class
メンバ関数"名前"がクラスの外側で再宣言されました。
- C5393 (E) Pointer to incomplete class type is not allowed
不完全クラスへのポインタ型は許されません。
- C5394 (E) Reference to local variable of enclosing function is not allowed
ローカルクラスを囲む関数のローカル変数へのリファレンスは許されません。
- C5397 (E) Implicitly generated assignment operator cannot copy:
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。
- C5399 (I) Entity-kind "名前" has an operator newxxxx() but no default
operator deletexxxx()
"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。
- C5400 (I) Entity-kind "名前" has a default operator deletexxxx() but no
operator newxxxx()
"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。
- C5401 (E) Destructor for base class "型" is not virtual
基底クラス"型"のデストラクタが virtual ではありません。
- C5403 (E) Entity-kind "名前" has already been declared
メンバ関数"名前"が再宣言されています。
- C5404 (E) Function "main" may not be declared inline
main 関数を inline 宣言することはできません。
- C5405 (E) Member function with the same name as its class must be a
constructor
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。
- C5407 (E) A destructor may not have parameters
デストラクタは引数を持つことができません。

- C5408 (E) Copy constructor for class "型 1" may not have a parameter of type "型 2"
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。
- C5409 (E) Entity-kind "名前" returns incomplete type "型"
関数"名前"のリターン型が不完全型"型"です。
- C5410 (E) Protected entity-kind "名前" is not accessible through a "型" pointer or object
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。
- C5411 (E) A parameter is not allowed
仮引数は許されません。
- C5412 (E) An "asm" declaration is not allowed here
asm 宣言は許されません。
- C5413 (E) No suitable conversion function from "型 1" to "型 2" exists
"型 1"から"型 2"への適切な変換関数が存在しません。
- C5414 (W) Delete of pointer to incomplete class
不完全型クラスへのポインタは削除されました。
- C5415 (E) No suitable constructor exists to convert from "型 1" to "型 2"
"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。
- C5416 (E) More than one constructor applies to convert from "型 1" to "型 2":
"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。
- C5417 (E) More than one conversion function from "型 1" to "型 2" applies:
"型 1"から"型 2"への変換関数があいまいです。
- C5418 (E) More than one conversion function from "型" to a built-in type applies:
"型"からビルトイン型への変換関数があいまいです。
- C5424 (E) A constructor or destructor may not have its address taken
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。
- C5427 (E) Qualified name is not allowed in member declaration
限定名をメンバ宣言のなかで使用できません。
- C5429 (E) The size of an array in "new" must be non-negative
new で指定された配列のサイズに負の値は許されません。

12. コンパイラのエラーメッセージ

- C5430 (W) Returning reference to local temporary
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。
- C5432 (E) "enum" declaration is not allowed
enum 型宣言は許されません。
- C5433 (E) Qualifiers dropped in binding reference of type "型 1" to
initializer of type "型 2"
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。
- C5434 (E) A reference of type "型 1" (not const-qualified) cannot be
initialized with a value of type "型 2"
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。
- C5435 (E) A pointer to function may not be deleted
関数へのポインタを削除することはできません。
- C5436 (E) Conversion function must be a nonstatic member function
変換関数は非静的メンバ関数でなければなりません。
- C5437 (E) Template declaration is not allowed here
このスコープ内でテンプレート宣言は許されません。
- C5438 (E) Expected a "<"
"<"が必要です。
- C5439 (E) Expected a ">"
">"が必要です。
- C5440 (E) Template parameter declaration is missing
テンプレートの引数宣言が正しくありません。
- C5441 (E) Argument list for entity-kind "名前" is missing
テンプレート"名前"の実引数リストが正しくありません。
- C5442 (E) Too few arguments for entity-kind "名前"
テンプレート"名前"の実引数が足りません。
- C5443 (E) Too many arguments for entity-kind "名前"
テンプレートの実引数が多すぎます。
- C5445 (E) Entity-kind "名前 1" is not used in declaring the parameter types
of entity-kind "名前 2"
テンプレート"名前 1"の引数"名前 2"が使用されません。

- C5449 (E) More than one instance of entity-kind "名前" matches the required type
オーバーロード関数"名前"があいまいです。
- C5452 (E) Return type may not be specified on a conversion function
変換関数のリターン型が指定されていません。
- C5456 (E) Excessive recursion at instantiation of entity-kind "名前"
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。
- C5457 (E) "名前" is not a function or static data member
"名前"が関数または静的データメンバではありません。
- C5458 (E) Argument of type "型1" is incompatible with template parameter
of type "型2"
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。
- C5459 (E) Initialization requiring a temporary or conversion is not allowed
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。
- C5461 (E) Initial value of reference to non-const must be an lvalue
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。
- C5463 (E) "template" is not allowed
"template"指定は許されません。
- C5464 (E) "型" is not a class template
"型"がクラステンプレートではありません。
- C5466 (E) "main" is not a valid name for a function template
"main"は関数テンプレートの名前に使用できません。
- C5467 (E) Invalid reference to entity-kind "名前" (union/nonunion mismatch)
"名前"の参照が不正です。
- C5468 (E) A template argument may not reference a local type
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。
- C5469 (E) Tag kind of "名前1" is incompatible with declaration of
entity-kind "名前2" (declared at line "行番号")
タグ名"名前1"の種類と"名前2"の宣言が合致しません。
- C5470 (E) The global scope has no tag named "名前"
グローバルスコープにタグ名"名前"がありません。
- C5471 (E) Entity-kind "名前1" has no tag member named "名前2"
"名前1"はタグメンバ"名前2"を持ちません。

12. コンパイラのエラーメッセージ

- C5473 (E) Entity-kind "名前" may be used only in pointer-to-member declaration
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。
- C5475 (E) A template argument may not reference a non-external entity
テンプレートの実引数は外部名以外を参照できません。
- C5476 (E) Name followed by "::~" must be a class name or a type name
::~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。
- C5477 (E) Destructor name does not match name of class "型"
クラス名"型"とデストラクタ名が合致しません。
- C5478 (E) Type used as destructor name does not match type "型"
デストラクタ名で使われた型と"型"が合致しません。
- C5479 (I) Entity-kind "名前" redeclared "inline" after being called
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。
- C5481 (E) Invalid storage class for a template declaration
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。
- C5484 (E) Invalid explicit instantiation declaration
テンプレートの実引数が不正です。
- C5485 (E) Entity-kind "名前" is not an entity that can be instantiated
テンプレート"名前"を実体化できません。
- C5486 (E) Compiler generated entity-kind "entity" cannot be explicitly instantiated
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。
- C5487 (E) Inline entity-kind "名前" cannot be explicitly instantiated
インライン関数"名前"を実体化することはできません。
- C5488 (E) Pure virtual entity-kind "名前" cannot be explicitly instantiated
純粋仮想関数"名前"を実体化することはできません。
- C5489 (E) Entity-kind "名前" cannot be instantiated -- no template definition was supplied
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。
- C5490 (E) Entity-kind "名前" cannot be instantiated -- it has been explicitly specialized
"名前"を実体化することはできません。

- C5493 (E) No instance of entity-kind "名前" matches the specified type
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。
- C5496 (E) Template parameter "名前" may not be redeclared in this scope
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。
- C5497 (W) Declaration of "名前" hides template parameter
"名前"の宣言はテンプレート引数を隠蔽します。
- C5498 (E) Template argument list must match the parameter list
テンプレート実引数と仮引数が合致しません。
- C5499 (E) Conversion function to convert from "型 1" to "型 2" is not allowed
"型 1"から"型 2"への変換関数は許されません。
- C5500 (E) Extra parameter of postfix "operatorxxxxx" must be of type "int"
後置演算関数の第2パラメタの型は int 型でなければなりません。
- C5501 (E) An operator name must be declared as a function
演算子名は関数として宣言しなければなりません。
- C5502 (E) Operator name is not allowed
演算子名は許されません。
- C5503 (E) Entity-kind "名前" cannot be specialized in the current scope
スコープ内で"名前"があいまいです。
- C5505 (E) Too few template parameters -- does not match previous declaration
テンプレートの引数が足りません。
- C5506 (E) Too many template parameters -- does not match previous declaration
テンプレートの引数が多すぎます。
- C5507 (E) Function template for operator delete(void *) is not allowed
operator delete(void *)の関数テンプレートは許されません。
- C5508 (E) Class template and template parameter may not have the same name
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。
- C5510 (E) A template argument may not reference an unnamed type
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。
- C5511 (E) Enumerated type is not allowed
enum 型は許されません。
- C5512 (W) Type qualifier on a reference type is not allowed
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。

12. コンパイラのエラーメッセージ

- C5513 (E) A value of type "型 1" cannot be assigned to an entity of type "型 2"
型不一致のため"型 1"の値を"型 2"の実体に代入することができません。
- C5514 (W) Pointless comparison of unsigned integer with a negative constant
負の定数と符号無し整数を比較しています。
- C5515 (E) Cannot convert to incomplete class "型"
不完全型"型"への型変換はできません。
- C5516 (E) Const object requires an initializer
const 型のオブジェクトには初期値が必要です。
- C5517 (E) Object has an uninitialized const or reference member
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。
- C5519 (E) Entity-kind "名前" may not have a template argument list
"名前"はテンプレート実引数を持つことができません。
- C5520 (E) Initialization with "{...}" expected for aggregate object
集成型のオブジェクトは{...}の形式で初期化しなければなりません。
- C5521 (E) Pointer-to-member selection class types are incompatible ("型 1" and "型 2")
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。
- C5522 (W) Pointless friend declaration
自分自身へのフレンド宣言をしています。
- C5526 (E) A parameter may not have void type
void 型の引数は指定できません。
- C5529 (E) This operator is not allowed in a template argument expression
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。
- C5530 (E) Try block requires at least one handler
try 文に対応する catch 文がありません。
- C5531 (E) Handler requires an exception declaration
catch 文の(...)には例外宣言が必要です。
- C5532 (E) Handler is masked by default handler
デフォルトハンドラによってハンドラがマスクされました。
- C5533 (E) Handler is potentially masked by previous handler for type "型"
"型"を持つ前のハンドラによってハンドラがマスクされる可能性があります。

- C5534 (I) Use of a local type to specify an exception
ローカルな型を使用した例外処理が指定されています。
- C5535 (I) Redundant type in exception specification
例外処理中に冗長な型の指定があります。
- C5536 (E) Exception specification is incompatible with that of previous
entity-kind "名前" (declared at line "行番号"):
例外処理指定が前の指定"名前"と合致しません。
- C5540 (E) Support for exception handling is disabled
例外処理を行うオプション(exception)が指定されていません。
- C5541 (W) Omission of exception specification is incompatible with previous
entity-kind "名前" (declared at line "行番号")
例外処理の省略形が前の"名前"と合致しません。
- C5542 (F) Could not create instantiation request file "名前"
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。
- C5543 (E) Non-arithmetic operation not allowed in nontype template argument
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。
- C5544 (E) Use of a local type to declare a nonlocal variable
ローカルでない変数にローカルな型を指定しています。
- C5545 (E) Use of a local type to declare a function
関数宣言にローカルな型を指定しています。
- C5546 (E) Transfer of control bypasses initialization of:
初期化処理が行われません。
- C5548 (E) Transfer of control into an exception handler
例外ハンドラ処理が実行されます。
- C5549 (I) Entity-kind "名前" is used before its value is set
"名前"に値を設定する前に使用しています。
- C5550 (W) Entity-kind "名前" was set but never used
"名前"が使用されませんでした。
- C5551 (E) Entity-kind "名前" cannot be defined in the current scope
"名前"はこのスコープ内で定義できません。
- C5552 (W) Exception specification is not allowed
例外処理指定は許されません。例外処理を無効にします。

12. コンパイラのエラーメッセージ

- C5553 (W) External/internal linkage conflict for entity-kind "名前"
(declared at line "行番号")
"名前"の外部/内部リンケージ指定が衝突します。外部リンケージを設定します。
- C5554 (W) Entity-kind "名前" will not be called for implicit or explicit conversions
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。
- C5555 (E) Tag kind of "名前" is incompatible with template parameter of type "型"
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。
- C5556 (E) Function template for operator new(size_t) is not allowed
operator new(size_t)の関数テンプレートは許されません。
- C5558 (E) Pointer to member of type "型" is not allowed
メンバへのポインタ型"型"が誤っています。
- C5559 (E) Ellipsis is not allowed in operator function parameter list
省略指定(...)は演算子関数の引数リストに指定できません。
- C5598 (E) A template parameter may not have void type
テンプレートの引数に void 型は指定できません。
- C5601 (E) A throw expression may not have void type
throw 式に void 型は指定できません。
- C5603 (E) Parameter of abstract class type "型" is not allowed:
抽象クラス"型"の引数は許されません。
- C5604 (E) Array of abstract class "型" is not allowed:
抽象クラス"型"の配列は許されません。
- C5610 (W) Entity-kind "名前 1" does not match "名前 2" -- virtual function override intended?
"名前 1"と"名前 2"が一致しません。指定された通りに処理を続けます。
- C5611 (W) Overloaded virtual function "名前 1" is only partially overridden in entity-kind "名前 2"
"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続けます。
- C5612 (E) Specific definition of inline template function must precede its first use
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。

- C5624 (E) "名前" is not a type name
"名前"は型の名前ではありません。
- C5641 (F) "名前" is not a valid directory
"名前"が正しいディレクトリではありません。
- C5642 (F) Cannot build temporary file name
コンパイラが使用するテンポラリファイルを作成できません。
- C5656 (E) Transfer of control into a try block
外側のブロックから try ブロックに制御が移ります。
- C5657 (W) Inline specification is incompatible with previous "名前"
(declared at line "行番号")
インライン指定が前の宣言"名前"と合致しません。
- C5658 (E) Closing brace of template definition not found
テンプレート定義の閉じ括弧がありません。
- C5662 (W) Call of pure virtual function
純粋仮想関数が関数を呼び出しています。
- C5663 (E) Invalid source file identifier string
#pragma 指定の構文に誤りがあります。
- C5664 (E) A class template cannot be defined in a friend declaration
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。
- C5673 (E) A reference of type "型 1" cannot be initialized with a value of
type "型 2"
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。
- C5674 (E) Initial value of reference to const volatile must be an lvalue
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。
- C5678 (I) Call of entity-kind "名前" (declared at line "行番号") cannot be
inlined
関数呼び出し"名前"がインライン展開されませんでした。
- C5679 (I) Entity-kind "名前" cannot be inlined
関数"名前"はインライン展開されません。
- C5693 (E) <typeinfo> must be included before typeid is used
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。
- C5694 (E) "名前" cannot cast away const or other type qualifiers
"名前"のキャストの結果 const などの属性がなくなります。

12. コンパイラのエラーメッセージ

- C5695 (E) The type in a `dynamic_cast` must be a pointer or reference to a complete class type, or `void *`
`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。
- C5696 (E) The operand of a pointer `dynamic_cast` must be a pointer to a complete class type
`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。
- C5697 (E) The operand of a reference `dynamic_cast` must be an lvalue of a complete class type
`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。
- C5698 (E) The operand of a runtime `dynamic_cast` must have a polymorphic class type
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。
- C5701 (E) An array type is not allowed here
配列型は許されません。
- C5702 (E) Expected an "="
代入式が必要です。
- C5703 (E) Expected a declarator in condition declaration
宣言子が必要です。
- C5704 (E) "名前", declared in condition, may not be redeclared in this scope
このスコープ内で"名前"を再宣言することはできません。
- C5705 (E) Default template arguments are not allowed for function templates
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。
- C5706 (E) Expected a ",", or ">"
", "または">"が必要です。
- C5707 (E) Expected a template parameter list
テンプレートの引数リストが必要です。
- C5708 (W) Incrementing a `bool` value is deprecated
`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続けます。
- C5709 (E) `bool` type is not allowed
`bool` 型の値をデクリメントすることはできません。

- C5710 (E) Offset of base class "名前 1" within class "名前 2" is too large
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。
- C5711 (E) Expression must have bool type (or be convertible to bool)
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。
- C5717 (E) The type in a const_cast must be a pointer, reference, or pointer
to member to an object type
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。
- C5718 (E) A const_cast can only adjust type qualifiers; it cannot change
the underlying type
const_cast は const/volatile 以外の型を調整することはできません。
- C5719 (E) mutable is not allowed
mutable の指定は許されません。
- C5720 (W) Redeclaration of entity-kind "名前" is not allowed to alter its
access
"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。
- C5722 (W) Use of alternative token "<:" appears to be unintended
2 文字表記 "<:" が使用されました。 "<" と解釈します。
- C5723 (W) Use of alternative token "%:" appears to be unintended
2 文字表記 "%:" が使用されました。 "%" と解釈します。
- C5724 (E) namespace definition is not allowed
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。
- C5725 (E) Name must be a namespace name
namespace の名前が正しくありません。
- C5726 (E) Namespace alias definition is not allowed
namespace の別名定義はここでは許されません。
- C5727 (E) namespace-qualified name is required
namespace の限定名が要求されます。
- C5728 (E) A namespace name is not allowed
namespace 名は許されません。
- C5730 (E) Entity-kind "名前" is not a class template
"名前"はクラステンプレートのメンバではありません。

12. コンパイラのエラーメッセージ

- C5732 (E) Allocation operator may not be declared in a namespace
operator new 関数が namespace 内で宣言されています。
- C5733 (E) Deallocation operator may not be declared in a namespace
operator delete 関数が namespace 内で宣言されています。
- C5734 (E) Entity-kind "名前 1" conflicts with using-declaration of
entity-kind "名前 2"
名前"名前 1"が using 宣言名"名前 2"と衝突します。
- C5735 (E) Using-declaration of entity-kind "名前 1" conflicts with
entity-kind "名前 2" (declared at line "行番号")
using 宣言の名前が衝突します。
- C5737 (W) Using-declaration ignored -- it refers to the current namespace
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。
- C5738 (E) A class-qualified name is required
クラスの限定名が要求されています。
- C5741 (W) Using-declaration of entity-kind "名前" ignored
using 宣言"名前"は無効です。
- C5742 (E) Entity-kind "名前 1" has no actual member "名前 2"
"名前 1"に"名前 2"のメンバは存在しません。
- C5750 (E) Entity-kind "名前" (declared at line "行番号") was used before its
template was declared
"名前"はテンプレートが宣言される前に使われました。
- C5751 (E) Static and nonstatic member functions with same parameter types
cannot be overloaded
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。
- C5752 (E) No prior declaration of entity-kind "名前"
namespace テンプレート関数"名前"の宣言がありません。
- C5753 (E) A template-id is not allowed
ここではテンプレート(template 名<template 実引数>)は許されません。
- C5754 (E) A class-qualified name is not allowed
ここではクラス限定名は許されません。
- C5755 (E) Entity-kind "名前" may not be redeclared in the current scope
このスコープ内で"名前"を再宣言することはできません。

- C5756 (E) Qualified name is not allowed in namespace member declaration
namespace メンバの宣言で指定された限定名は許されません。
- C5757 (E) Entity-kind "名前" is not a type name
"名前"は型名ではありません。
- C5761 (E) Typename may only be used within a template
typename キーワードはテンプレート内でのみ使用できます。
- C5766 (W) Exception specification for virtual entity-kind "名前 1" is
incompatible with that of overridden entity-kind "名前 2"
仮想関数の例外指定"名前 1"が"名前 2"に合致しません。
- C5767 (W) Conversion from pointer to smaller integer
ポインタをポインタサイズより小さい型に変換しています。
- C5768 (W) Exception specification for implicitly declared virtual
entity-kind "名前 1" is incompatible with that of overridden
entity-kind "名前 2"
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。
- C5771 (E) "explicit" is not allowed
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。
- C5772 (E) Declaration conflicts with "名前" (reserved class name)
予約されたクラス名 type_info と衝突します。
- C5773 (E) Only "()" is allowed as initializer for array entity-kind
"名前"
配列"名前"の初期化指定が正しくありません。
- C5774 (E) "virtual" is not allowed in a function template declaration
関数テンプレートに virtual 指定はできません。
- C5775 (E) Invalid anonymous union -- class member template is not allowed
無名 union の指定が正しくありません。
- C5776 (E) Template nesting depth does not match the previous declaration
of entity-kind "名前"
テンプレートのパラメタのネストが前の宣言"名前"と合致しません。
- C5777 (E) This declaration cannot have multiple "template <...>" clauses
この宣言に複数のテンプレート宣言はできません。
- C5779 (E) "名前", declared in for-loop initialization, may not be redeclared
in this scope
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。

12. コンパイラのエラーメッセージ

- C5782 (E) Definition of virtual entity-kind "名前" is required here
ここに仮想関数の定義"名前"が要求されます。
- C5784 (E) A storage class is not allowed in a friend declaration
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。
- C5785 (E) Template parameter list for "名前" is not allowed in this
declaration
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。
- C5786 (E) entity-kind "名前" is not a valid member class or function template
"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。
- C5787 (E) Not a valid member class or function template declaration
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。
- C5788 (E) A template declaration containing a template parameter list may
not be followed by an explicit specialization declaration
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。
- C5789 (E) Explicit specialization of entity-kind "名前 1" must precede the
first use of entity-kind "名前 2"
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。
- C5790 (E) Explicit specialization is not allowed in the current scope
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。
- C5791 (E) Partial specialization of entity-kind "名前" is not allowed
テンプレート"名前"の部分的な定義は許されません。
- C5792 (E) Entity-kind "名前" is not an entity that can be explicitly
specialized
"名前"はテンプレートのインスタンスではありません。
- C5793 (E) Explicit specialization of entity-kind "名前" must precede its
first use
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。
- C5794 (W) Template parameter "テンプレート引数" may not be used in an elaborated
type specifier
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。

- C5795 (E) Specializing entity-kind "名前" requires "template<>" syntax
"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。
- C5800 (E) This declaration may not have extern "C" linkage
この宣言は `extern "C"`リンケージを持つことはできません。
- C5801 (E) "名前" is not a class or function template name in the current scope
"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。
- C5802 (W) Specifying a default argument when redeclaring an unreferenced function template is nonstandard
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。
- C5803 (E) Specifying a default argument when redeclaring an already referenced function template is not allowed
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。
- C5804 (E) Cannot convert pointer to member of base class "型 1" to pointer to member of derived class "型 2" -- base class is virtual
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。
- C5805 (E) Exception specification is incompatible with that of entity-kind "名前" (declared at line "行番号"):
`throw` 例外指定は"名前"の例外指定と合致しません。
- C5806 (W) Omission of exception specification is incompatible with entity-kind "名前" (declared at line "行番号")
`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。
- C5807 (E) The parse of this expression has changed between the point at which it appeared in the program and the point at which the expression was evaluated -- "typename" may be required to resolve the ambiguity
デフォルト実引数の式の評価が途中で終了しました。
- C5808 (E) Default-initialization of reference is not allowed
リファレンス型のデフォルトの初期化は許されません。
- C5809 (E) Uninitialized entity-kind "名前" has a const member
未初期化の"名前"が `const` 型メンバを持ちます。
- C5810 (E) Uninitialized base class "型" has a const member
未初期化の基底クラス"型"が `const` 型メンバを持ちます。

12. コンパイラのエラーメッセージ

- C5811 (E) Const entity-kind "名前" requires an initializer -- class "型"
has no explicitly declared default constructor
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。
- C5812 (W) Const object requires an initializer -- class "型" has no explicitly
declared default constructor
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。
- C5815 (I) Type qualifier on return type is meaningless
テンプレートで実体化されるリターン型に意味のない修飾型を指定しています。
修飾型を有効にします。
- C5817 (E) Static data member declaration is not allowed in this class
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。
- C5818 (E) Template instantiation resulted in an invalid function
declaration
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。
- C5822 (E) Invalid destructor name for type "型"
"型"のデストラクタ名が正しくありません。
- C5824 (E) Destructor reference is ambiguous -- both entity-kind "名前 1"
and entity-kind "名前 2" could be used
"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。
- C5825 (W) Virtual inline entity-kind "名前" was never defined
仮想インラインメンバ関数"名前"の定義がありません。
- C5826 (W) Entity-kind "名前" was never referenced
関数の引数"名前"は参照されません。
- C5827 (E) Only one member of a union may be specified in a constructor
initializer list
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。
- C5831 (I) Support for placement delete is disabled
operator delete 関数の型が正しくありません。処理を継続します。
- C5832 (E) No appropriate operator delete is visible
適当な operator delete 関数が見つかりません。
- C5833 (E) Pointer or reference to incomplete type is not allowed
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。

- C5834 (E) Invalid partial specialization -- entity-kind "名前" is already fully specialized
すでに特別化された"名前"を部分特別化しています。
- C5835 (E) Incompatible exception specifications
例外指定の型が合致しません。
- C5836 (W) Returning reference to local variable
ローカル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。
- C5837 (W) Omission of explicit type is nonstandard ("int" assumed)
型指定がありません。int 型を仮定します。
- C5838 (E) More than one partial specialization matches the template argument list of entity-kind "名前"
部分特別化テンプレート"名前"のテンプレート実引数があいまいです。
- C5840 (E) A template argument list is not allowed in a declaration of a primary template
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。
- C5841 (E) Partial specializations may not have default template arguments
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。
- C5842 (E) Entity-kind "名前 1" is not used in template argument list of entity-kind "名前 2"
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。
- C5843 (E) The type of partial specialization template parameter entity-kind "名前" depends on another template parameter
部分特別化テンプレート"名前"のテンプレート仮引数が別のテンプレート仮引数に依存しています。
- C5844 (E) The template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter
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。
- C5845 (E) This partial specialization would have been used to instantiate entity-kind "名前"
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。
- C5846 (E) This partial specialization would have been made the instantiation of entity-kind "名前" ambiguous
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。

12. コンパイラのエラーメッセージ

- C5847 (E) Expression must have integral or enum type
式の型は整数型か enum 型でなければなりません。
- C5848 (E) Expression must have arithmetic or enum type
式の型は算術型か enum 型でなければなりません。
- C5849 (E) Expression must have arithmetic, enum, or pointer type
式の型は算術型、enum 型もしくはポインタ型でなければなりません。
- C5850 (E) Type of cast must be integral or enum
キャストの型は整数型か enum 型でなければなりません。
- C5851 (E) Type of cast must be arithmetic, enum, or pointer
キャストの型は算術型、enum 型もしくはポインタ型でなければなりません。
- C5852 (E) Expression must be a pointer to a complete object type
式の型は完全オブジェクト型へのポインタ型でなければなりません。
- C5853 (E) A partial specialization of a member class template must be declared in the class of which it is a member
メンバクラスの部分特別化テンプレートはそのメンバを含むクラスの中で宣言しなければなりません。
- C5854 (E) A partial specialization nontype argument must be the name of a nontype parameter or a constant
部分特別化テンプレートの非型テンプレート実引数は非型の仮引数名か定数でなければなりません。
- C5855 (E) Return type is not identical to return type "型" of overridden virtual function entity-kind "名前"
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"と同一ではありません。
- C5857 (E) A partial specialization of a class template must be declared in the namespace of which it is a member
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。
- C5858 (E) Entity-kind "名前" is a pure virtual function
"名前"は純粋仮想関数です。
- C5859 (E) Pure virtual entity-kind "名前" has no overrider
純粋仮想関数"名前"はオーバーライドされません。
- C5861 (E) Invalid character in input line
行中に不正な文字が現れました。

- C5862 (E) Function returns incomplete type "型"
関数のリターン型"型"が不完全型です。
- C5864 (E) "名前" is not a template
"名前"はテンプレートではありません。
- C5865 (E) A friend declaration may not declare a partial specialization
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。
- C5867 (W) Declaration of "size_t" does not match the expected type "型"
size_t 型が期待する"型"と異なります。
- C5868 (E) Space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator)
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。
- C5870 (E) Invalid multibyte character sequence
不正な 2 バイト文字があります。
- C5871 (E) 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")
"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。
- C5873 (E) Non-integral operation not allowed in nontype template argument
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。
- C5875 (E) Embedded C++ does not support templates
Embedded C++仕様はテンプレート機能をサポートしません。
- C5876 (E) Embedded C++ does not support exception handling
Embedded C++仕様は例外処理機能をサポートしません。
- C5877 (E) Embedded C++ does not support namespaces
Embedded C++仕様は namespace 機能をサポートしません。
- C5878 (E) Embedded C++ does not support run-time type information
Embedded C++仕様はランタイム型情報機能をサポートしません。
- C5879 (E) Embedded C++ does not support the new cast syntax
Embedded C++仕様は new のキャスト機能をサポートしません。
- C5880 (E) Embedded C++ does not support using-declarations
Embedded C++仕様は using 宣言機能をサポートしません。
- C5881 (E) Embedded C++ does not support "mutable"
Embedded C++仕様は mutable 機能をサポートしません。

12. コンパイラのエラーメッセージ

- C5882 (E) Embedded C++ does not support multiple or virtual inheritance
Embedded C++仕様は多重継承/仮想継承機能をサポートしません。
- C5885 (E) "型 1" cannot be used to designate constructor for "型 2"
"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。
- C5891 (E) An explicit template argument list is not allowed on this
declaration
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。
- C5894 (E) Entity-kind "名前" is not a template
"名前"はテンプレートではありません。
- C5896 (E) Expected a template argument
テンプレートの実引数が期待されます。
- C5898 (E) Nonmember operator requires a parameter with class or enum type
非メンバ演算子関数にはクラスまたは enum 型の仮引数が要求されます。
- C5900 (E) Using-declaration of entity-kind "名前" is not allowed
"名前"の using 宣言は許されません。
- C5901 (E) Qualifier of destructor name "型 1" does not match type "型 2"
"型 1"のデストラクタの限定名が"型 2"に一致しません。
- C5902 (W) Type qualifier ignored
型限定名が不正です。型限定名を無効にします。
- C5916 (E) Cannot convert pointer to member of derived class "型 1" to pointer
to member of base class "型 2" -- base class is virtual
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。
- C5919 (F) Invalid output file: "名前"
テンプレート情報ファイルの"名前"が不正です。
コンパイラの実環境設定やホスト環境のファイルシステム異常がないか確認ください。
- C5920 (F) Cannot open output file: "名前"
テンプレート情報ファイル"名前"をオープンすることができません。
コンパイラの実環境設定やホスト環境のファイルシステム異常がないか確認ください。
- C5926 (F) Cannot open definition list file: "名前"
ファイル"名前"をオープンすることができません。
コンパイラの実環境設定やホスト環境のファイルシステム異常がないか確認ください。
- C5928 (E) Incorrect use of va_start
va_start の使用方法に誤りがあります。

- C5929 (E) Incorrect use of `va_arg`
`va_arg` の使用方法に誤りがあります。
- C5930 (E) Incorrect use of `va_end`
`va_end` の使用方法に誤りがあります。
- C5935 (E) "typedef" may not be specified here
typedef を指定することはできません。
- C5936 (W) Redeclaration of entity-kind "名前" alters its access
"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。
- C5937 (E) A class or namespace qualified name is required
クラスまたは namespace の限定名が要求されます。
- C5940 (W) Missing return statement at end of non-void entity-kind "名前"
void 型以外をリターンする関数"名前"が return 文を持ちません。
return 値は不定になります。
- C5941 (W) Duplicate using-declaration of "名前" ignored
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。
- C5946 (E) Name following "template" must be a member template
"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。
- C5947 (E) Name following "template" must have a template argument list
"template"に続く名前はテンプレート実引数でなければなりません。
- C5952 (E) A template parameter may not have class type
テンプレート仮引数にクラス型名は指定できません。
- C5953 (E) A default template argument cannot be specified on the declaration
of a member of a class template
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。
- C5954 (E) A return statement is not allowed in a handler of a function try
block of a constructor
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。
- C5959 (W) Declared size for bit field is larger than the size of the bit
field type; truncated to "サイズ" bits
指定されたビット数がビットフィールドの型の"サイズ"を超えています。
ビット数をビットフィールドの型のサイズに合わせて処理を続けます。
- C5960 (E) Type used as constructor name does not match type "型"
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。

12. コンパイラのエラーメッセージ

- C5961 (W) Use of a type with no linkage to declare a variable with linkage
リンケージを持たない型を使用してリンケージを持つ変数として宣言しています。
リンケージを持つものとします。
- C5962 (W) Use of a type with no linkage to declare a function
リンケージを持たない型を使用してリンケージを持つ関数として宣言しています。
リンケージを持つものとします。
- C5963 (E) Return type may not be specified on a constructor
コンストラクタにリターン型を指定できません。
- C5964 (E) Return type may not be specified on a destructor
デストラクタにリターン型を指定できません。
- C5965 (E) Incorrectly formed universal character name
universal character の形式が正しくありません。
- C5966 (E) Universal character name specifies an invalid character
universal character で指定された文字が不正です。
- C5967 (E) A universal character name cannot designate a character in the
basic character set
基本文字集合内で universal character を文字として指定することはできません。
- C5968 (E) This universal character is not allowed in an identifier
識別子にこの universal character は許されません。
- C5978 (E) A template friend declaration cannot be declared in a local class
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。
- C5979 (E) Ambiguous "?" operation: second operand of type "型 1" can be
converted to third operand type "型 2", and vice versa
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。
- C5980 (E) Call of an object of a class type without appropriate operator()
or conversion functions to pointer-to-function type
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。
- C5982 (E) There is more than one way an object of type "型" can be called
for the argument list
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。
- C5988 (E) Invalid pragma declaration
#pragma の構文が不正です。

C5989 (E) "名前" has already been specified by other pragma
このシンボルは既に他の#pragma 指定がされています。

C5990 (E) Pragma may not be specified after definition
シンボル定義後の宣言にのみ#pragma 指定することはできません。

C5991 (E) Invalid kind of pragma is specified to this symbol
不正な#pragma を指定しました。

12.3 標準ライブラリ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されており、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
1100 (ERANGE)	Data out of range オーバーフローが発生しました。	atan, cos, sin, tan, cosh, sinh, tanh, exp, fabs, frexp, ldexp, modf, ceil, floor, strtol, atoi, fscanf, scanf, sscanf, atol
1101 (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, asin, atan2, log, log10, sqrt, fmod, pow
1102 (EDIV)	Division by zero ゼロによる除算を行っています。	divbs, divws, divls, divbu, divwu, divlu
1104 (ESTRN)	Too long string 文字列の長さが 512 文字を超えています。	strtol, strtod, atof, atoi, atol
1106 (PTRERR)	Invalid file pointer ファイルポインタの値に NULL ポインタ定数を指定しています。	fclose, fflush, freopen, setbuf, setvbuf, fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, fgetc, fgets, fputc, fputs, ungetc, fread, fwrite, fseek, ftell, rewind, perror
1200 (ECBASE)	Invalid radix 基数の指定が誤っています。	strtol, atol, atoi
1202 (ETLN)	Number too long 数値を表現する文字列の長さが 17 桁を超えています。	strtod, fscanf, scanf, sscanf, atof
1204 (EEXP)	Exponent too large 指数部の桁数が 3 桁を超えています。	strtod, fscanf, scanf, sscanf, atof
1206 (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。	strtod, fscanf, sscanf, atof
1210 (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)。	strtod, fscanf, scanf, sscanf, atof
1220 (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(アンダフロー)。	strtod, fscanf, scanf, sscanf, atof
1250 (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)。	strtod, fscanf, scanf, sscanf, atof
1260 (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(アンダフロー)。	strtod, fscanf, scanf, sscanf, atof
1270 (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
1280 (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(アンダフロー)。	fscanf, scanf, sscanf

12. コンパイラのエラーメッセージ

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
1300 (NOTOPN)	File not open ファイルがオープンされていません。	fclose, fflush, setbuf, setvbuf, fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, fgetc, fgets, fputc, fputs, gets, puts, ungetc, fread, fwrite, fseek, ftell, rewind, perror, freopen
1302 (EBADF)	Bad file number 入力専用ファイルに対して出力関数、あるいは出力専用ファイルに対して入力関数を発行しています。	fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, fgetc, fgets, fputc, fputs, gets, puts, ungetc, perror, fread, fwrite
1304 (ECSPEC)	Error in format 書式付き入出力関数で指定している書式が誤っています。	fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, perror
2100 (EMALRESM)	Error in waiting semaphore malloc 用セマフォ資源確保に失敗しました。	free, malloc, realloc
2101 (EMALFRSM)	Error in signaling semaphore malloc 用セマフォ資源解放に失敗しました。	free, malloc, realloc
2110 (ETOKRESM)	Error in waiting semaphore strtok 用セマフォ資源確保に失敗しました。	strtok
2111 (ETOKFRSM)	Error in signaling semaphore strtok 用セマフォ資源解放に失敗しました。	strtok
2120 (EIOBRESM)	Error in waiting semaphore _job 用セマフォ資源確保に失敗しました。	fopen
2121 (EIOBFRSM)	Error in signaling semaphore _job 用セマフォ資源解放に失敗しました。	fopen

13. アセンブラのエラーメッセージ

13.1 エラー形式とエラーレベル

本章では、以下の形式で出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ
エラー内容

エラーレベルは、エラーの重要度に従い、3種類に分類されます。

エラーレベル	動作
(W) ウォーニング	処理を継続します。
(E) エラー	処理を継続します。
(F) フェータル	処理を中断します。

13.2 メッセージ一覧

- 10 (E) NO INPUT FILE SPECIFIED
入力ソースファイルの指定がありません。
入力ソースファイルを指定してください。
- 20 (E) CANNOT OPEN FILE ファイル名
指定のファイルをオープンできません。
ファイル名、ディレクトリ名などを見直してください。
- 30 (E) INVALID COMMAND PARAMETER
オプションに誤りがあります。
オプションを見直してください。
- 40 (E) CANNOT ALLOCATE MEMORY
処理中にメモリが足りなくなりました。
ユーザが使用できるメモリ量が極端に少ない場合に発生します。
他に実行中の処理があればその処理を終了してからアセンブラを再起動してください。
それでも本エラーが発生する場合はホストシステムのメモリ管理の方法を見直してください。
- 50 (E) INVALID FILE NAME ファイル名
ディレクトリを含めたファイル名が長すぎるか、ファイル名に誤りがあります。
ファイル名を見直してください。
このときアセンブラが出力するオブジェクトモジュールはデバッガで扱えない可能性があります。

13. アセンブラのエラーメッセージ

- 101 (E) SYNTAX ERROR IN SOURCE STATEMENT
ソースステートメントに構文上の誤りがあります。
ソースステートメント全体を見直してください。
- 102 (E) SYNTAX ERROR IN DIRECTIVE
アセンブラ制御命令のソースステートメントに構文上の誤りがあります。
ソースステートメント全体を見直してください。
- 104 (E) LOCATION COUNTER OVERFLOW
ロケーションカウンタ値が最大値を超えています。
プログラムを縮小してください。
- 105 (E) ILLEGAL INSTRUCTION IN STACK SECTION
スタックセクション内に実行命令、DSP 命令、拡張命令、データを確保するアセンブラ制御命令を記述しています。
実行命令、DSP 命令、拡張命令、データを確保するアセンブラ制御命令を削除してください。
- 106 (E) TOO MANY ERRORS
エラーの数が多いので表示を打ち切りました。
ソースステートメント全体を見直してください。
- 108 (E) ILLEGAL CONTINUATION LINE
複数行にわたって記述したソースステートメントに誤りがあります。
記述方法を見直してください。
- 150 (E) INVALID DELAY SLOT INSTRUCTION
ディレイスロット命令(遅延分岐命令の直後にくる実行命令)が不当です。
実行命令を記述する順序を変更するなどして、ディレイスロット命令が不当とならないようにしてください。
- 151 (E) ILLEGAL EXTENDED INSTRUCTION POSITION
ディレイスロット命令に拡張命令を記述しています。
ディレイスロット命令には実行命令を記述してください。
- 152 (E) ILLEGAL BOUNDARY ALIGNMENT VALUE
拡張命令を記述しているセクションの境界調整数が不当です。
拡張命令を記述するセクションの境界調整数は 2 以上を指定してください。
- 160 (E) REPEAT LOOP NESTING
REPEAT 拡張命令～終了アドレス間に別の REPEAT 拡張命令が存在します。
REPEAT 拡張命令の位置を訂正してください。
- 161 (E) ILLEGAL START ADDRESS FOR REPEAT LOOP
REPEAT 拡張命令～開始アドレス間に実行命令または DSP 命令が存在しません。
REPEAT 拡張命令と開始アドレスの間に 1 つ以上の実行命令または DSP 命令を記述してください。

- 162 (E) ILLEGAL DATA BEFORE REPEAT LOOP
REPEAT 拡張命令で指定したループの直前に不当なデータが存在します。
ループ直前がアセンブラ制御命令の場合、アセンブラ制御命令を訂正してください。
ループ直前がリテラルプールの場合、.NOPPOOL でリテラルプールの出力を抑止してください。
リピートする命令が 3 命令以下の場合、ループの直前は実行命令または DSP 命令でなければなりません。
- 200 (E) UNDEFINED SYMBOL REFERENCE
参照しているシンボルが定義されていません。
シンボルを定義してください。
- 201 (E) ILLEGAL SYMBOL OR SECTION NAME
シンボル(セクション名を含む)として予約語を指定しています。
シンボル(セクション名を含む)を訂正してください。
- 202 (E) ILLEGAL SYMBOL OR SECTION NAME
シンボル(セクション名を含む)に誤りがあります。
シンボル(セクション名を含む)を訂正してください。
- 203 (E) ILLEGAL LOCAL LABEL
ローカルラベルの指定に誤りがあります。
ローカルラベルの指定を訂正してください。
- 300 (E) ILLEGAL MNEMONIC
オペレーションに誤りがあります。
オペレーションを訂正してください。
- 301 (E) TOO MANY OPERANDS OR ILLEGAL COMMENT
実行命令のオペランドが多すぎるか、コメントに誤りがあります。
オペランドまたはコメントを訂正してください。
- 304 (E) LACKING OPERANDS
オペランドが足りません。
オペランドを訂正してください。
- 307 (E) ILLEGAL ADDRESSING MODE
オペランドに許されないアドレス形式を指定しています。
オペランドを訂正してください。
- 308 (E) SYNTAX ERROR IN OPERAND
オペランドに文法上の誤りがあります。
オペランドを訂正してください。

13. アセンブラのエラーメッセージ

- 309 (E) FLOATING POINT REGISTER MISMATCH
単精度のオペレーションに対して倍精度の浮動小数点レジスタを指定したか、倍精度のオペレーションに対して単精度浮動小数点レジスタを指定しています。
オペレーションサイズまたは浮動小数点レジスタを訂正してください。
- 350 (E) SYNTAX ERROR IN SOURCE STATEMENT (ニーモニック)
DSP 命令のソースステートメントに構文上の誤りがあります。
ソースステートメントを見直してください。
- 351 (E) ILLEGAL COMBINATION OF MNEMONICS (ニーモニック, ニーモニック)
許されていない DSP 演算命令の組み合わせを指定しています。
命令の組み合わせを訂正してください。
- 352 (E) ILLEGAL CONDITION (ニーモニック)
DSP 演算命令に対するコンディションを不正に指定しています。
コンディションの指定を削除するか、DSP 演算命令を変更してください。
- 353 (E) ILLEGAL POSITION OF INSTRUCTION (ニーモニック)
DSP 演算命令の指定位置が誤っています。
DSP 演算命令を正しい位置に指定してください。
- 354 (E) ILLEGAL ADDRESSING MODE (ニーモニック)
DSP 演算命令のアドレス形式が誤っています。
オペランドを訂正してください。
- 355 (E) ILLEGAL REGISTER NAME (ニーモニック)
DSP 演算命令のレジスタ指定に誤りがあります。
レジスタ名を訂正してください。
- 357 (E) ILLEGAL COMBINATION OF MNEMONICS (ニーモニック)
データ転送命令の指定が不正です。
データ転送命令を訂正してください。
- 371 (E) ILLEGAL COMBINATION OF MNEMONICS (ニーモニック, ニーモニック)
データ転送命令の組み合わせが誤っています。
データ転送命令の組み合わせを訂正してください。
- 372 (E) ILLEGAL ADDRESSING MODE (ニーモニック)
データ転送命令のオペランドとして許されないアドレス形式を指定しています。
オペランドを訂正してください。
- 373 (E) ILLEGAL REGISTER NAME (ニーモニック)
データ転送命令のレジスタの指定が誤っています。
レジスタ名を訂正してください。

- 400 (E) CHARACTER CONSTANT TOO LONG
文字定数が 4 文字を超えています。
文字定数を訂正してください。
- 402 (E) ILLEGAL VALUE IN OPERAND
オペランドとして範囲外の値です。
値を変更してください。
- 403 (E) ILLEGAL OPERATION FOR RELATIVE VALUE
相対アドレスに対して乗除算または論理演算を指定しています。
演算内容を訂正してください。
- 406 (E) ILLEGAL OPERAND
浮動小数点データを指定するところに式を指定しています。
式ではなく浮動小数点定数を指定してください。
- 407 (E) MEMORY OVERFLOW
式の計算中、計算用のメモリが足りなくなりました。
演算内容を簡単にしてください。
- 408 (E) DIVISION BY ZERO
0 除算を指定しています。
演算内容を変更してください。
- 409 (E) REGISTER IN EXPRESSION
式の中にレジスタ名が現れました。
演算内容を訂正してください。
- 411 (E) INVALID STARTOF/SIZEOF OPERAND
STARTOF 演算または SIZEOF 演算で誤ったセクション名を指定しています。
セクション名を訂正してください。
- 412 (E) ILLEGAL SYMBOL IN EXPRESSION
シフト数に相対アドレスを指定しています。
演算内容を訂正してください。
- 450 (E) ILLEGAL DISPLACEMENT VALUE
ディスプレイメント値が不当です(負の値を指定しています)。
ディスプレイメント値を訂正してください。
- 452 (E) ILLEGAL DATA AREA ADDRESS
PC 相対データ転送命令のデータ領域のアドレスが不当です。
各命令のオペレーションサイズに合った境界のアドレスにアクセスしてください。
(MOV.L, MOVA 命令は 4 バイト境界、MOV.W は 2 バイト境界です)

13. アセンブラのエラーメッセージ

- 453 (E) LITERAL POOL OVERFLOW
リテラル未出力の拡張命令が 510 個を超えています。
.POOL を用いるなどしてリテラルプールを出力してください。
- 460 (E) ILLEGAL SYMBOL
REPEAT 拡張命令のオペランドに前方参照でないラベル、未定義シンボル、ラベル以外のシンボルを指定したか、開始アドレスが終了アドレスより大きな値になっています。
オペランドを訂正してください。
- 461 (E) SYNTAX ERROR IN OPERAND
不当なオペランドを指定しています。
オペランドを訂正してください。
- 462 (E) ILLEGAL VALUE IN OPERAND
REPEAT 拡張命令とラベルの距離が範囲外です。
REPEAT 拡張命令またはラベルの位置を訂正してください。
- 463 (E) NO INSTRUCTION IN REPEAT LOOP
リピートループ内に命令が存在しないか、終了アドレスに指定した位置に命令が存在しません。
開始アドレス～終了アドレス間に命令を記述するか、命令の存在するアドレスを終了アドレスに指定してください。
- 500 (E) SYMBOL NOT FOUND
ラベルが必要なアセンブラ制御命令のソースステートメントにラベルがありません。
ラベルを記述してください。
- 501 (E) ILLEGAL ADDRESS VALUE IN OPERAND
セクションの先頭アドレスの指定が誤っているか、ロケーションカウンタ値の指定が誤っています。
先頭アドレスまたはロケーションカウンタ値を訂正してください。
- 502 (E) ILLEGAL SYMBOL IN OPERAND
オペランドに不当な値(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、未定義シンボル)を指定しています。
オペランドを訂正してください。
- 503 (E) UNDEFINED EXPORT SYMBOL
ファイル内で定義していないシンボルを外部定義シンボルとして宣言しています。
シンボルを定義するか、外部定義シンボルとしての宣言を取りやめてください。
- 504 (E) INVALID RELATIVE SYMBOL IN OPERAND
オペランドに不当な値(前方参照シンボル、外部参照シンボル)を指定しています。
オペランドを訂正してください。

- 505 (E) ILLEGAL OPERAND
オペランドの名称に誤りがあります。
オペランドを訂正してください。
- 506 (E) ILLEGAL OPERAND
オペランドとして許されない要素を指定しています。
オペランドを訂正してください。
- 508 (E) ILLEGAL VALUE IN OPERAND
オペランドに範囲外の値を指定しています。
オペランドを訂正してください。
- 510 (E) ILLEGAL BOUNDARY VALUE
境界調整数の指定に誤りがあります。
境界調整数を訂正してください。
- 512 (E) ILLEGAL EXECUTION START ADDRESS
実行開始アドレスに誤りがあります。
実行開始アドレスを訂正してください。
- 513 (E) ILLEGAL REGISTER NAME
レジスタ名に誤りがあります。
レジスタ名を訂正してください。
- 514 (E) INVALID EXPORT SYMBOL
外部定義できないシンボルを外部定義シンボルとして宣言しています。
外部定義シンボルとしての宣言を取りやめてください。
- 516 (E) EXCLUSIVE DIRECTIVES
制御命令の指定内容が矛盾しています。
関連する制御命令を含めて見直してください。
- 517 (E) INVALID VALUE IN OPERAND
オペランドに不当な値(前方参照シンボル、外部参照シンボル、他セクションの相対アドレスシンボル)を指定しています。
オペランドを訂正してください。
- 518 (E) INVALID IMPORT SYMBOL
ファイル内で定義しているシンボルを外部参照シンボルとして宣言しています。
外部参照シンボルとしての宣言を取りやめてください。
- 520 (E) ILLEGAL .CPU DIRECTIVE POSITION
.CPU がプログラムの先頭にないか、複数回指定しています。
.CPU はプログラムの先頭に 1 回だけ指定してください。

13. アセンブラのエラーメッセージ

- 521 (E) ILLEGAL .NOPOOL DIRECTIVE POSITION
.NOPOOL の記述位置が不当です。
.NOPOOL はディレイスロット命令に続けて記述してください。
- 522 (E) ILLEGAL .POOL DIRECTIVE POSITION
.POOL を遅延分岐命令に続けて記述しています。
遅延分岐命令の後にはディレイスロット命令を記述してください。
- 523 (E) ILLEGAL OPERAND
.LINE のオペランドに誤りがあります。
.LINE のオペランドを訂正してください。
- 525 (E) ILLEGAL .LINE DIRECTIVE POSITION
.LINE をマクロ展開または条件付き繰り返し展開内に指定しています。
.LINE の指定位置を変えてください。
- 526 (E) STRING TOO LONG
オペランドの文字列が 255 文字を超えています。
.SDATA、.SDATAB、.SDATAC、.SDATAZ のオペランドに指定する文字列は 255 文字以内としてください。
- 527 (E) CANNOT SUPPORT COMMON SECTION SINCE VERSION 5
セクション属性に COMMON を指定しています。
コモンセクションは使用できなくなりました。
最適化リンカージェネレータの start オプションでコロン(:)を用いて複数セクションを同一アドレスに配置できます。
- 528 (E) SPECIFICATION OF THE ADDRESS OVERLAPS
セクション内のアドレス割付けが重複しています。
.SECTION、.ORG の指定内容を見直してください。
- 529 (E) THE ADDRESS BETWEEN SECTIONS OVERLAPS
セクション間のアドレス割付けが重複しています。
.SECTION、.ORG の指定内容を見直してください。
- 530 (E) ILLEGAL OPERAND
.FILE のオペランドに誤りがあります。
.FILE のオペランドを訂正してください。
- 531 (E) ILLEGAL .FILE DIRECTIVE POSITION
.FILE をマクロ展開または条件付き繰り返し展開内に指定しています。
.FILE の指定位置を変えてください。
- 600 (E) INVALID CHARACTER
ソースプログラムに不当な文字があります。
不当な文字を訂正してください。

- 601 (E) INVALID DELIMITER
区切り文字が不当です。
区切り文字を訂正してください。
- 602 (E) INVALID CHARACTER STRING FORMAT
文字列に誤りがあります。
文字列を訂正してください。
- 603 (E) SYNTAX ERROR IN SOURCE STATEMENT
ソースステートメントに構文上の誤りがあります。
ソースステートメント全体を見直してください。
- 604 (E) ILLEGAL SYMBOL IN OPERAND
制御命令のオペランドが不当です。
本制御命令のオペランドにシンボルおよびロケーションカウンタ(\$)は記述できません。
- 610 (E) MULTIPLE MACRO NAMES
.MACRO で定義しようとしているマクロ命令は既に定義されています。
マクロ名を訂正してください。
- 611 (E) MACRO NAME NOT FOUND
.MACRO のオペランドにマクロ名がありません。
マクロ名を記述してください。
- 612 (E) ILLEGAL MACRO NAME
.MACRO のマクロ名に誤りがあります。
マクロ名を訂正してください。
- 613 (E) ILLEGAL .MACRO DIRECTIVE POSITION
マクロ本体(.MACRO ~ .ENDM 間)、.AREPEAT ~ .AENDR 間、.AWHILE ~ .AENDW 間に .MACRO があります。
.MACRO を削除してください。
- 614 (E) MULTIPLE MACRO PARAMETERS
マクロ定義(.MACRO)の仮引数の宣言で仮引数名が重複しています。
仮引数名を訂正してください。
- 615 (E) ILLEGAL .END DIRECTIVE POSITION
マクロ本体(.MACRO ~ .ENDM 間)に .END があります。
.END を削除してください。
- 616 (E) MACRO DIRECTIVES MISMATCH
.ENDM が .MACRO に対応していないか、.EXITM がマクロ本体(.MACRO ~ .ENDM 間)、.AREPEAT ~ .AENDR 間、.AWHILE ~ .AENDW 間以外にあります。
.ENDM または .EXITM を削除してください。

13. アセンブラのエラーメッセージ

- 618 (E) MACRO EXPANSION TOO LONG
マクロ展開で 1 行の文字数が 8,192 文字を超えています。
8,192 文字以下になるように訂正してください。
- 619 (E) ILLEGAL MACRO PARAMETER
マクロコールでマクロパラメータの仮引数名に誤りがあるか、マクロ本体(.MACRO
~.ENDM 間)の仮引数名に誤りがあります。
仮引数名を訂正してください。
マクロ本体の仮引数名が誤りの場合はマクロ展開時にエラーになります。
- 620 (E) UNDEFINED PREPROCESSOR VARIABLE
参照しているプリプロセッサ変数が定義されていません。
プリプロセッサ変数を定義してください。
- 621 (E) ILLEGAL .END DIRECTIVE POSITION
マクロ展開中に .END があります。
.END を削除してください。
- 622 (E) ') ' NOT FOUND
マクロ処理除外の閉じカッコがありません。
マクロ処理除外の閉じカッコを記述してください。
- 623 (E) SYNTAX ERROR IN STRING FUNCTION
文字列操作関数に構文上の誤りがあります。
文字列操作関数を見直してください。
- 624 (E) MACRO PARAMETERS MISMATCH
マクロコールで位置指定のマクロパラメータの数が多すぎます。
マクロパラメータの数を訂正してください。
- 631 (E) END DIRECTIVE MISMATCH
対になる制御文で終了の制御文が一致しません。
制御文を見直してください。
- 640 (E) SYNTAX ERROR IN OPERAND
条件つきアセンブリ制御文のオペランドに構文上の誤りがあります。
ソースステートメント全体を見直してください。
- 641 (E) INVALID RELATIONAL OPERATOR
条件つきアセンブリ制御文のオペランドの関係演算子に誤りがあります。
関係演算子を訂正してください。
- 642 (E) ILLEGAL .END DIRECTIVE POSITION
.AREPEAT ~ .AENDR 間、.AWHILE ~ .AENDW 間に .END があります。
.END を削除してください。

- 643 (E) DIRECTIVE MISMATCH
.AREPEAT、.AWHILE に対する .AENDR、.AENDW が対になっていません。
制御文を見直してください。
- 644 (E) ILLEGAL .AENDW OR .AENDR DIRECTIVE POSITION
.AIF ~ .AENDI 間に .AENDR、.AENDW があります。
.AENDR、.AENDW を削除してください。
- 645 (E) EXPANSION TOO LONG
.AREPEAT、.AWHILE 展開で 1 行の文字数が 8,192 文字を超えています。
8,192 文字以下になるように訂正してください。
- 650 (E) INVALID INCLUDE FILE
.INCLUDE のファイル名に誤りがあります。
ファイル名を訂正してください。
- 651 (E) CANNOT OPEN INCLUDE FILE
.INCLUDE のファイルをオープンできません。
ファイル名を訂正してください。
- 652 (E) INCLUDE NEST TOO DEEP
ファイルインクルードのネストが 30 レベルを超えています。
ネストを 30 レベル以下にしてください。
- 653 (E) SYNTAX ERROR IN OPERAND
.INCLUDE のオペランドに構文上の誤りがあります。
オペランドを訂正してください。
- 660 (E) .ENDM NOT FOUND
.MACRO に対する .ENDM がありません。
.ENDM を記述してください。
- 662 (E) ILLEGAL .END DIRECTIVE POSITION
.AIF ~ .AENDI 間に .END があります。
.END を削除してください。
- 663 (E) ILLEGAL .END DIRECTIVE POSITION
インクルードファイル中に .END があります。
.END を削除してください。
- 664 (E) ILLEGAL .END DIRECTIVE POSITION
.AIF ~ .AENDI 間に .END があります。
.END を削除してください。
- 665 (E) EXPANSION TOO LONG
.DEFINE で 1 行の文字数が 8,192 文字を超えています。
8,192 文字以下になるように訂正してください。

13. アセンブラのエラーメッセージ

- 667 (E) SUCCESSFUL CONDITION .AERROR
.AERROR を含むステートメントが .AIF の条件によって処理されています。
.AERROR を処理しないように条件式を見直してください。
- 668 (E) ILLEGAL VALUE IN OPERAND
.AIFDEF のオペランドに誤りがあります。
本制御命令のオペランドは .DEFINE のシンボルで指定してください。
- 669 (E) STRING TOO LONG
オペランドの文字列が 255 文字を超えています。
.ASSIGNC、.DEFINE、文字列操作関数(.LEN、.INSTR、.SUBSTR)のオペランドに指定する文字列は 255 文字以内としてください。
- 700 (W) ILLEGAL VALUE IN OPERAND (ニーモニック)
DSP 演算命令のオペランドが値の範囲を超えています。
値を訂正してください。
- 701 (W) MULTIPLE REGISTER IN DESTINATION (ニーモニック, ニーモニック)
DSP 演算命令のデスティネーションオペランドに複数の同一レジスタを指定しています。
レジスタの指定を訂正してください。
- 702 (W) ILLEGAL OPERATION SIZE (ニーモニック)
DSP 演算命令またはデータ転送命令のオペレーションサイズが誤っています。
オペレーションサイズを訂正または削除してください。
- 703 (W) MULTIPLE REGISTER IN DESTINATION (ニーモニック, ニーモニック)
DSP 演算命令とデータ転送命令で同一のデスティネーションレジスタを指定しています。
レジスタの指定を訂正してください。
- 800 (W) SYMBOL NAME TOO LONG
プリプロセッサ変数または置換シンボルが 32 文字を超えています。
プリプロセッサ変数または置換シンボルを訂正してください。
アセンブラは 33 文字目以降を無視します。
- 801 (W) MULTIPLE SYMBOLS
定義済みのシンボルを再び定義しています。
シンボルの再定義を取りやめてください。
アセンブラは 2 度目以降の定義を無視します。
- 807 (W) ILLEGAL OPERATION SIZE
オペレーションサイズに誤りがあります。
オペレーションサイズを訂正してください。
アセンブラはオペレーションサイズの指定を無視します。

- 808 (W) ILLEGAL CONSTANT SIZE
整数定数の記述の一部に誤りがあります。
記述を訂正してください。
アセンブラが整数定数を誤って(プログラムの意図しない値として)解釈する可能性があります。
- 810 (W) TOO MANY OPERANDS
アセンブラ制御命令のオペランドが多すぎるか、コメントに誤りがあります。
オペランドまたはコメントを訂正してください。
アセンブラは余分なオペランドの指定を無視します。
- 811 (W) ILLEGAL SYMBOL DEFINITION
ラベルを記述できないアセンブラ制御命令のソースステートメントにラベルを記述しています。
ラベルを削除してください。
アセンブラはラベルを無視します。
- 813 (W) SECTION ATTRIBUTE MISMATCH
セクションの再開で異なる種類のセクションを指定しているか、絶対アドレスセクションの再開でセクションの先頭アドレスを再び指定しています。
セクションを再開する場合はセクションの種類や先頭アドレスを指定しないでください。
セクションを開始したときの指定がそのまま有効です。
- 815 (W) MULTIPLE MODULE NAMES
オブジェクトモジュール名を再設定しています。
オブジェクトモジュールの設定は1度だけにしてください。
アセンブラは2度目以降の設定を無視します。
- 816 (W) ILLEGAL DATA AREA ADDRESS
データまたはデータ領域の配置が不当です。
ワード単位のデータやデータ領域は先頭アドレスが偶数になるように確保してください。
ロングワードまたは単精度単位のデータやデータ領域は先頭アドレスが4の倍数になるように確保してください。
倍精度単位のデータやデータ領域は先頭アドレスが8の倍数になるように確保してください。
アセンブラはデータまたはデータ領域を指定どおりに配置します。
- 817 (W) ILLEGAL BOUNDARY VALUE
コードセクションの境界調整数が4未満です。
指定は有効です。
ただし、実行命令、DSP命令、拡張命令を奇数アドレスに記述している場合はウォーニング882になります。
コードセクションの境界調整数に1を指定する場合は特に注意してください。

13. アセンブラのエラーメッセージ

- 818 (W) COMMANDLINE OPTION MISMATCH FOR FLOATING DIRECTIVE
CPU 種別が SH2E のとき、round=nearest または denormalize=on を指定していません。
round オプションまたは denormalize オプションの指定を変更してください。
アセンブラは round オプションまたは denormalize オプションの指定どおりにオブジェクトコードを生成します。
- 825 (W) ILLEGAL INSTRUCTION IN DUMMY SECTION
ダミーセクションに実行命令、DSP 命令、拡張命令、データを確保するアセンブラ制御命令を記述しています。
実行命令、DSP 命令、拡張命令、データを確保するアセンブラ制御命令を削除してください。
アセンブラは実行命令、拡張命令、DSP 命令、データを確保するアセンブラ制御命令の記述を無視します。
- 826 (W) ILLEGAL PRECISION
浮動小数点定数の精度がオペレーションサイズで指定した精度と異なります。
オペレーションサイズで指定した精度または浮動小数点定数の精度を訂正してください。
アセンブラはオペレーションサイズで指定した精度を有効として処理します。
- 832 (W) MULTIPLE 'P' DEFINITIONS
デフォルトセクション名としての P が他のシンボルである P と重複しています。
P が重複しないようにしてください。
アセンブラは P をデフォルトセクション名とみなし、他のシンボル P の定義を無効とします。
- 835 (W) ILLEGAL VALUE IN OPERAND
実行命令のオペランドに範囲外の値を指定しています。
値を訂正してください。
アセンブラは値を範囲内に補正してオブジェクトコードを生成します。
- 836 (W) ILLEGAL CONSTANT SIZE
整数定数の記述の一部に誤りがあります。
記述を訂正してください。
アセンブラが整数定数を誤って(プログラムの意図しない値として)解釈する可能性があります。
- 837 (W) SOURCE STATEMENT TOO LONG
ソースステートメントの 1 行の長さが 8,192 バイトを超えています。
コメント文を短くするなどして 1 行を 8,192 バイト以内に納めてください。
または、ソースステートメントを複数行に分けて記述してください。

- 838 (W) ILLEGAL CHARACTER CODE
コメント、文字列以外にシフト JIS、EUC または LATIN1 コードを指定したか、sjis オプション、euc オプション、または latin1 オプションの指定がありません。
シフト JIS コード、EUC コードまたは LATIN1 コードはコメント、文字列内に指定してください。もしくは、sjis オプション、euc オプション、latin1 オプションを指定してください。
- 839 (W) ILLEGAL FIGURE IN OPERAND
固定小数点データでワードサイズの場合は 6 桁以上、ロングワードサイズの場合は 11 桁以上のデータを指定しました。
余分な桁を削除してください。
- 840 (W) OPERAND OVERFLOW
浮動小数点データがオーバフローしました。
値を変更してください。
正の値の場合は+、負の値の場合は- になります。
- 841 (W) OPERAND UNDERFLOW
浮動小数点データがアンダフローしました。
値を変更してください。
正の値の場合は+0、負の値の場合は-0 になります。
- 842 (W) OPERAND DENORMALIZED
浮動小数点データに非正規化数を指定しました。
浮動小数点データを確認してください。
アセンブラは指定どおりにオブジェクトコードを生成します。
(非正規化数を設定します)
- 850 (W) ILLEGAL SYMBOL DEFINITION
ラベルフィールドにシンボルを指定しました。
シンボルを削除してください。
- 851 (W) MACRO SERIAL NUMBER OVERFLOW
マクロ生成番号が 99,999 を超えています。
マクロコールの回数を減らしてください。
- 852 (W) UNNECESSARY CHARACTER
オペランドの終了後に文字があります。
オペランドを訂正してください。
- 854 (W) .AWHILE ABORTED BY .ALIMIT
展開回数が .ALIMIT で設定した上限値に達したため展開を中断しました。
繰り返しを展開する条件を見直してください。

13. アセンブラのエラーメッセージ

- 870 (W) ILLEGAL DISPLACEMENT VALUE
ディスペースメント値が不当です。
オペレーションサイズがワードのときにディスペースメントが偶数になっていません。
オペレーションサイズがロングワードのときにディスペースメントが4の倍数になっていません。
アセンブラがディスペースメントを補正することを考慮してください。
アセンブラはオペレーションサイズに応じてディスペースメントを補正してオブジェクトコードを生成します。
オペレーションサイズがワードのときはディスペースメントが偶数になるように切り捨てます。
オペレーションサイズがロングワードのときはディスペースメントが4の倍数になるように切り捨てます。
- 871 (W) PC RELATIVE IN DELAY SLOT
PC 相対アドレス形式の実行命令がメモリ上で遅延分岐命令の直後に位置しています。
遅延分岐によって PC 値が変化することを考慮してください。
アセンブラは指定どおりにオブジェクトコードを生成します。
- 874 (W) CANNOT CHECK DATA AREA BOUNDARY
PC 相対データ転送命令において、データ領域の境界をチェックできません。
リンクする際にはデータ領域の境界に十分注意してください。
データ転送命令が相対アドレスセクションに属している場合や、データ領域が外部参照シンボルである場合などにアセンブラは本ウォーニングを出力します。
- 875 (W) CANNOT CHECK DISPLACEMENT SIZE
PC 相対データ転送命令において、ディスペースメントの大きさをチェックできません。リンクする際にはデータ転送命令とデータ領域の距離に十分注意してください。
データ転送命令が相対アドレスセクションに属している場合や、データ領域が外部参照シンボルである場合などにアセンブラは本ウォーニングを出力します。
- 876 (W) ASSEMBLER OUTPUTS BRA INSTRUCTION
アセンブラが自動的に BRA 命令を出力しました。
.POOL などを用いてリテラルプールの出力位置を指定するか、アセンブラが自動的に生成した BRA 命令でプログラムが正常に動作するかどうかを確認してください。
リテラルプールの出力ポイントが見つからないため、リテラルプールとそれを飛び越すための BRA 命令を自動出力したことを示します。
- 880 (W) .END NOT FOUND
プログラムに .END がありません。
.END を記述してください
- 881 (W) ILLEGAL DIRECTIVE IN REPEAT LOOP
リピートループ内に不正なアセンブラ制御命令を指定しています。
不正なアセンブラ制御命令を削除してください。
リピートループ内にデータまたはデータ領域を確保する制御命令、.ALIGN、.ORG を記述した場合、1 制御命令を 1 命令としてリピートする命令数をカウントします。

- 882 (W) ILLEGAL ADDRESS
実行命令、拡張命令を奇数アドレスに記述しています。
実行命令、拡張命令は偶数アドレスに記述してください。
- 883 (W) MULTIPLE FILE NAMES
.FILE によるファイル指定が複数回あります。
2 回目以降の指定を無視します。
- 884 (W) "CPU 種別 1" IS INTERPRETED AS " CPU 種別 2"
cpu=<CPU 種別 1>は無効です。
cpu=<CPU 種別 2>と解釈してアセンブルします。
- 885 (W) CANNOT SUPPORT "CPU 種別"
.CPU で指定した CPU 種別はサポートしていません。
cpu オプションで指定した CPU 種別として解釈します。
- 901 (F) SOURCE FILE INPUT ERROR
ソースファイルの入力時にエラーが発生しました。
ディスクの空き容量を確認してください。
ディスク上の不要なファイルを削除するなどして必要な空き容量を確保してください。
- 902 (F) MEMORY OVERFLOW
メモリ不足です(中間語に関する情報を処理できません)。
プログラムを分割してください。
- 903 (F) LISTING FILE OUTPUT ERROR
リストファイルの出力時にエラーが発生しました。
ディスクの空き容量を確認してください。
ディスク上の不要なファイルを削除するなどして必要な空き容量を確保してください。
- 904 (F) OBJECT FILE OUTPUT ERROR
オブジェクトファイルの出力時にエラーが発生しました。
ディスクの空き容量を確認してください。
ディスク上の不要なファイルを削除するなどして必要な空き容量を確保してください。
- 905 (F) MEMORY OVERFLOW
メモリ不足です(ソースプログラムの行に関する情報を処理できません)。
プログラムを分割してください。
- 906 (F) MEMORY OVERFLOW
メモリ不足です(シンボルに関する情報を処理できません)。
プログラムを分割してください。
- 907 (F) MEMORY OVERFLOW
メモリ不足です(セクションに関する情報を処理できません)。
プログラムを分割してください。

13. アセンブラのエラーメッセージ

- 908 (F) SECTION OVERFLOW
セクションの個数が多すぎます。
デバッグ情報を出力するときは 62,265 までです。
デバッグ情報を出力しないときは 65,274 個までです。
プログラムを分割してください。
- 933 (F) ILLEGAL ENVIRONMENT VARIABLE
CPU 種別に誤りがあります。
CPU 種別を訂正してください。
- 935 (F) SUBCOMMAND FILE INPUT ERROR
サブコマンドファイル入力時にエラーが発生しました。
ディスクの空き容量を確認してください。
ディスク上の不要なファイルを削除するなどして必要な空き容量を確保してください。
- 950 (F) MEMORY OVERFLOW
メモリ不足です。
ソースプログラムを分割してください。
- 951 (F) LITERAL POOL OVERFLOW
リテラルプール用の内部シンボルの個数が 100,000 個を超えています。
ソースプログラムを分割してください。
- 952 (F) LITERAL POOL OVERFLOW
リテラルプールの容量があふれています。
リテラルプールがあふれる前に無条件分岐を挿入してください。
- 953 (F) MEMORY OVERFLOW
メモリ不足です。
ソースプログラムを分割してください。
- 954 (F) LOCAL BLOCK NUMBER OVERFLOW
ローカルラベルの有効範囲であるローカルブロックの個数が 100,000 個を超えていま
す。
ソースプログラムを分割してください。
- 956 (F) EXPAND FILE INPUT/OUTPUT ERROR
プリプロセッサ展開出力のファイル出力時にエラーが発生しました。
ディスクの空き容量を確認してください。
ディスク上の不要なファイルを削除するなどして必要な空き容量を確保してください。
- 957 (F) MEMORY OVERFLOW
メモリ不足です。
ソースプログラムを分割してください。

- 958 (F) MEMORY OVERFLOW
メモリ不足です。
ソースプログラムを分割してください。
- 964 (F) MEMORY OVERFLOW
メモリ不足です(シンボルに関する情報を処理できません)。
ソースプログラムを分割してください。
- 970 (F) MEMORY OVERFLOW
メモリ不足です(セクションのサイズが大きすぎます)。
.ORG でロケーションカウンタに大きなオフセットを与えたり、.DATAB 等で大きなデータ領域を確保した可能性があります。
セクションを分割するか、データ領域を小さくしてください。

14. 最適化リンケージエディタのエラーメッセージ

14.1 エラー形式とエラーレベル

本章では、以下の形式で出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ
 エラー内容

エラーレベルは、エラーの重要度に従い、5 種類に分類されます。

	エラーレベル	動作
L0000 - L0999 P0000 - P0999	(I) インフォメーション	処理を継続します。
L1000 - L1999 P1000 - P1999	(W) ウォーニング	処理を継続します。
L2000 - L2999 P2000 - P2999	(E) エラー	オプション解析処理を継続し、処理を中断します。
L3000 - L3999 P3000 - P3999	(F) フェータル	処理を中断します。
L4000 - P4000 -	(-) インターナル	処理を中断します。

L で始まるエラー番号は、最適化リンケージエディタ出力メッセージです。

P で始まるエラー番号は、プレリンカ出力メッセージです。P で始まるエラー番号は、nomessage オプションや change_message オプションで指定できません。

14.2 メッセージ一覧

L0001 (I) Section "セクション" created by optimization "最適化"
"最適化"の最適化によって、"セクション"を作成しました。

L0002 (I) Symbol "シンボル" created by optimization "最適化"
"最適化"の最適化によって、"シンボル"を作成しました。

L0003 (I) "ファイル"- "シンボル" moved to "セクション" by optimization
variable_access の最適化によって、"ファイル"内の"シンボル"を移動しました。

L0004 (I) "ファイル"- "シンボル" deleted by optimization
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。

14. 最適化リンケージエディタのエラーメッセージ

- L0005 (I)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"**
"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。
- L0100 (I)No inter-module optimization information in "ファイル"
"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。ただし、asmsh には goptimize オプションはありません。
- L0101 (I)No stack information in "ファイル"
"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または SYSROF->ELF コンバートファイルの可能性があります。最適化リンケージエディタ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。
- P0200 (I)"インスタンス" no longer needed in "ファイル"
使用しない"インスタンス"が"ファイル"内にあります。
- P0201 (I)"インスタンス" assigned to file "ファイル"
"インスタンス"を"ファイル"に割り当てます。
- P0202 (I)Executing: "コマンド"
インスタンス生成のために"コマンド"を実行しています。
- P0203 (I)"インスタンス" adopted by file "ファイル"
"インスタンス"が"ファイル"に割り当てられました。
- L1000 (W)Option "オプション" ignored
"オプション"は無効です。"オプション"を無視します。
- L1001 (W)Option "オプション 1" is ineffective without option "オプション 2"
"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。
- L1002 (W)Option "オプション 1" cannot be combined with option "オプション 2"
"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。
- L1003 (W)Divided output file cannot be combined with option "オプション"
"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。
- L1004 (W)Fatal level message cannot be changed to other level : "番号"
Fatal レベルメッセージはレベル変更できません。"番号"の指定を無視します。change_message オプションで変更できるエラーは、Information/Warning/Error レベルです。

- L1005 (W)Subcommand file terminated with end option instead of exit option
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。
- L1006 (W)Options following exit option ignored
exit オプションの後のオプションを無視しました。
- L1007 (W)Duplicate option : "オプション"
"オプション"が重複しています。最後に指定した方を有効にします。
- L1010 (W)Duplicate file specified in option "オプション" : "ファイル"
"オプション"で同じファイルを2度指定しました。2度目の指定を無視します。
- L1011 (W)Duplicate module specified in option "オプション" : "モジュール"
"オプション"で同じモジュールを2度指定しました。2度目の指定を無視します。
- L1012 (W)Duplicate symbol/section specified in option "オプション" : "名前"
"オプション"で同じシンボル名またはセクション名を2度指定しました。2度目の指定を無視します。
- L1013 (W)Duplicate number specified in option "オプション" : "番号"
"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。
- L1100 (W)Cannot find "名前" specified in option "オプション"
"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。
- L1101 (W)"名前" in rename option conflicts between symbol and section
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。
- L1102 (W)Symbol "シンボル" redefined in option "オプション"
"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。
- L1103 (W)Invalid address value specified in option "オプション" : "アドレス"
"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。
- L1110 (W)Entry symbol "シンボル" in entry option conflicts
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。
- L1120 (W)Section address is not assigned to "セクション"
"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。
- L1121 (W)Address cannot be assigned to absolute section "セクション"
in start option
"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。

14. 最適化リンケージエディタのエラーメッセージ

- L1122 (W)Section address in start option is incompatible with alignment :
"セクション"
start オプションで指定した"セクション"のアドレスは境界調整数と矛盾しています。
境界調整数に合わせてセクションアドレスを補正します。
- L1130 (W)Section attribute mismatch in rom option :
"セクション 1, セクション 2"
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、境界調整数が異なります。
"セクション 2"の境界調整数はどちらか大きい方を有効とします。
- L1140 (W)Load address overflowed out of record-type in option "オプション"
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 形式を超える範囲は、別の record 形式で出力します。
- L1150 (W)Sections in fsymbol option have no symbol
fsymbol 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。fsymbol オプションを無視します。
- L1160 (W)Undefined external symbol "シンボル"
未定義の"シンボル"を参照しています。
- L1200 (W)Backed up file "ファイル 1" into "ファイル 2"
"ファイル 1"を"ファイル 2"にバックアップしました。
- L1300 (W)No debug information in input files
入力ファイル内にデバッグ情報がありません。debug オプションを無視します。コンパイラ、アセンブル時に debug オプションを指定しているか確認してください。
- L1301 (W)No inter-module optimization information in input files
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。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。
- L1302 (W)No stack information in input files
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または SYSROF->ELF コンバートファイルの場合は、stack オプションは無効です。
- L1310 (W)"セクション" in "ファイル" is not supported in this tool
"ファイル"内に未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。
- L1311 (W)Invalid debug information format in "ファイル"
"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。
- L1320 (W)Duplicate symbol "シンボル" in "ファイル"
"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。

- L1321 (W)Entry symbol "シンボル" in "ファイル" conflicts
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シンボルを有効にします。
- L1322 (W)Section alignment mismatch : "セクション"
境界調整数の異なる同名セクションを入力しました。境界調整数は最大の指定を有効にします。
- L1323 (W)Section attribute mismatch : "セクション"
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。
- L1324 (W)Symbol size mismatch : "シンボル" in "ファイル"
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。
- L1400 (W)Stack size overflow in register optimization
レジスタ最適化で、スタックアクセスコードがコンパイラのスタック量制限値を超えました。レジスタ最適化指定を無視します。
- L1401 (W)Function call nest too deep
関数の呼び出しネストが深すぎるため、レジスタ最適化を実施できません。
- L1410 (W)Cannot optimize "ファイル"-"セクション" due to
multi label relocation operation
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。
- L1420 (W)"ファイル" is newer than "プロファイル"
"ファイル"は"プロファイル"より後に更新されました。プロファイル情報を無視します。
- L1500 (W)Cannot check stack size
スタックセクションがないため、コンパイル時の stack オプションで指定したスタックサイズの整合性をチェックできません。コンパイル時の stack オプションの整合性をチェックするためにはコンパイル時、アセンブル時に goptimize オプション指定が必要です。
- L1501 (W)Stack size overflow : "スタックサイズ"
スタックセクションサイズが、コンパイル時に stack オプションで指定した"スタックサイズ"を超えました。コンパイル時のオプションを変更するか、スタック量を削減できるようにプログラムを変更してください。
- L1502 (W)Stack size in "ファイル" conflicts with that in another file
複数のファイルで異なるスタックサイズを指定されています。コンパイル時のオプションを確認してください。

14. 最適化リンケージエディタのエラーメッセージ

- P1600 (W)An error occurred during name decoding of "インスタンス"
"インスタンス"は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。
- L2000 (E)Invalid option : "オプション"
"オプション"はサポートしていません。
- L2001 (E)Option "オプション" cannot be specified on command line
"オプション"はコマンドライン上では指定できません。サブコマンドファイル内で指定してください。
- L2002 (E)Input option cannot be specified on command line
コマンドライン上で input オプションを指定しました。コマンドライン上での入力ファイル指定は input オプション無しで指定してください。
- L2003 (E)Subcommand option cannot be specified in subcommand file
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。
- L2004 (E)Option "オプション 1" cannot be combined with option "オプション 2"
"オプション 1"と"オプション 2"は同時に指定できません。
- L2005 (E)Option "オプション" cannot be specified while processing "プロセス"
"プロセス"処理に対して"オプション"は指定できません。
- L2006 (E)Option "オプション 1" is ineffective without option "オプション 2"
"オプション 1"は"オプション 2"が必要です。
- L2010 (E)Option "オプション" requires parameter
"オプション"はパラメタ指定が必要です。
- L2011 (E)Invalid parameter specified in option "オプション" : "パラメタ"
"オプション"で無効なパラメタを指定しました。
- L2012 (E)Invalid number specified in option "オプション" : "値"
"オプション"指定で無効な値を指定しました。値の範囲を確認してください。
- L2013 (E)Invalid address value specified in option "オプション" : "アドレス"
"オプション"で指定した"アドレス"は無効な値です。0 ~ FFFFFFFF の間の 16 進数で指定してください。
- L2014 (E)Illegal symbol/section name specified in "オプション" : "名前"
"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。セクション/シンボル名で使用できるのは数字、英字、_、\$(先頭は数字以外)です。
- L2020 (E)Duplicate file specified in option "オプション" : "ファイル"
"オプション"指定で同じファイルを 2 度指定しました。

- L2021 (E) Duplicate symbol/section specified in option "オプション" : "名前"
"オプション"指定で同じシンボル名またはセクション名を2度指定しました。
- L2022 (E) Address ranges overlap in option "オプション" : "アドレス範囲"
"オプション"で指定した"アドレス範囲"が重複しています。
- L2100 (E) Invalid address specified in cpu option : "アドレス"
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。
- L2101 (E) Invalid address specified in option "オプション" : "アドレス"
"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超えました。
- L2110 (E) Section size of second parameter in rom option is not 0 :
"セクション"
rom オプションの第2パラメタにサイズが0でない"セクション"を指定しました。
- L2111 (E) Absolute section cannot be specified in rom option : "セクション"
rom オプションで絶対アドレスセクションを指定しました。
- L2120 (E) Library "ファイル" without module name specified as input file
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。
- L2121 (E) Input file is not library file : "ファイル(モジュール)"
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。
- L2130 (E) Cannot find file specified in option "オプション" : "ファイル"
"オプション"で指定したファイルが見つかりません。
- L2131 (E) Cannot find module specified in option "オプション" : "モジュール"
"オプション"で指定したモジュールがありません。
- L2132 (E) Cannot find "名前" specified in option "オプション"
"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。
- L2133 (E) Cannot find defined symbol "名前" in option "オプション"
"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。
- L2140 (E) Symbol/section "名前" redefined in option "オプション"
"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。
- L2141 (E) Module "モジュール" redefined in option "オプション"
"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。
- L2200 (E) Illegal object file : "ファイル"
P2200 ELF フォーマット以外を入力しました。

14. 最適化リンケージエディタのエラーメッセージ

- L2201 (E)Illegal library file : "ファイル"
"ファイル"はライブラリファイルではありません。
- L2202 (E)Illegal cpu information file : "ファイル"
"ファイル"は cpu 情報ファイルではありません。
- L2203 (E)Illegal profile information file : "ファイル"
"ファイル"はプロファイル情報ファイルではありません。
- L2210 (E)Invalid input file type specified for option "オプション" :
"ファイル(種別)"
"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。
- L2211 (E)Invalid input file type specified while processing "プロセス" :
"ファイル(種別)"
"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。
- L2220 (E)Illegal mode type "モード種別" in "ファイル"
異なるモード種別のファイルを入力しました。
- L2221 (E)Section type mismatch : "セクション"
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。
- L2300 (E)Duplicate symbol "シンボル" in "ファイル"
"シンボル"は重複しています。
- L2301 (E)Duplicate module "モジュール" in "ファイル"
"モジュール"は重複しています。
- L2310 (E)Undefined external symbol "シンボル" referenced in "ファイル"
"ファイル"内で未定義の"シンボル"を参照しています。
- L2311 (E)Section "セクション 1" cannot refer to overlaid section :
"セクション 2"- "シンボル"
同一アドレスを指定したオーバーレイセクション間でシンボル参照がありました。
"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。
- L2320 (E)Section address overflowed out of range : "セクション"
"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。
- L2330 (E)Relocation size overflow : "ファイル"- "セクション"- "オフセット"
リロケーション演算結果がリロケーションサイズを超えました。ブランチ先が届かない、
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。
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。

- L2331 (E)Division by zero in relocation value calculation :
"ファイル"- "セクション"- "オフセット"
リロケーション演算に 0 除算が発生しました。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。
- L2332 (E)Relocation value is odd number :
"ファイル"- "セクション"- "オフセット"
リロケーション演算結果が奇数になりました。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。
- L2340 (E)Symbol name in section "セクション" is too long
fsymbol で指定した"セクション"内のシンボルの文字数が 8174 文字を超えました。
- L2400 (E)Global register in "ファイル" conflicts : "シンボル","レジスタ"
"ファイル"内で指定したグローバルレジスタにはすでに別のシンボルが割りついています。
- L2401 (E)__near8, __near16 symbol "シンボル" is outside near memory area
"シンボル"は__near8、__near16 の範囲に割りついていません。start 指定を変更するか、コンパイル時の__near 指定を外して、正しいアドレス計算ができるようにしてください。
- L2402 (E)Number of register parameter conflicts with that in another
file : "関数"
"関数"は複数のファイルで異なるレジスタパラメータ数を指定されています。
- L2410 (E)Address value specified by map file differs from one after linkage
as to "シンボル"
"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。
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ないか確認してください。または、optlnk 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、optlnk の最適化オプションを無効にしてください。
- L2411 (E)Map file in "ファイル" conflicts with that in another file
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。
- L2412 (E)Cannot open file : "ファイル"
"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。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。
- L2413 (E)Cannot close file : "ファイル"
"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。

14. 最適化リンケージエディタのエラーメッセージ

- L2414 (E)Cannot read file : "ファイル"
"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。
- L2415 (E)Illegal map file : "ファイル"
"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。ファイル名が正しいか確認してください。
- P2500 (E)Cannot find library file : "ファイル"
ライブラリとして指定した"ファイル"がありません。
- P2501 (E)"インスタンス" has been referenced as both an explicit specialization and a generated instantiation
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。
"インスタンス"を使用しているファイルに対して、form=relocate でリロケータブルオブジェクトファイルを作成していないか確認してください。
- P2502 (E)"インスタンス" assigned to "ファイル1" and "ファイル2"
"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。
"インスタンス"を使用しているファイルに対して、form=relocate でリロケータブルオブジェクトファイルを作成していないか確認してください。
- L3000 (F)No input file
入力ファイルがありません。
- L3001 (F)No module in library
ライブラリ内のモジュールが0になりました。
- L3002 (F)Option "オプション1" is ineffective without option "オプション2"
"オプション1"は"オプション2"が必要です。
- L3100 (F)Section address overflow out of range : "セクション"
"セクション"のアドレスがFFFFFFFFを超えました。start オプションのアドレス指定を変更してください。
- L3101 (F)Section "セクション1" overlaps section "セクション2"
"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。start オプションのアドレス指定を変更してください。
- L3102 (F)Section contents overlap in absolute section "セクション"
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。ソースプログラムを修正してください。
- L3110 (F)Illegal cpu type "cpu 種別" in "ファイル"
異なるcpu 種別のファイルを入力しました。

- L3111 (F)Illegal encode type "エンディアン種別" in "ファイル"
異なるエンディアン種別のファイルを入力しました。
- L3112 (F)Invalid relocation type in "ファイル"
"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。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。
- L3200 (F)Too many sections
セクション数が限界値を超えました。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。
- L3201 (F)Too many symbols
シンボル数が限界値を超えました。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。
- L3202 (F)Too many modules
モジュール数が限界値を超えました。ライブラリを分けて作成してください。
- L3300 (F)Cannot open file : "ファイル"
P3300 "ファイル"をオープンできません。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。
- L3301 (F)Cannot close file : "ファイル"
"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。
- L3302 (F)Cannot write file : "ファイル"
"ファイル"に書きこめません。ディスク容量に空きがない可能性があります。
- L3303 (F)Cannot read file : "ファイル"
P3303 "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。
- L3310 (F)Cannot open temporary file
P3310 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。
- L3311 (F)Cannot close temporary file
中間ファイルをクローズできません。ディスク容量に空きがない可能性があります。
- L3312 (F)Cannot write temporary file
中間ファイルに書きこめません。ディスク容量に空きがない可能性があります。
- L3313 (F)Cannot read temporary file
中間ファイルを読めません。ディスク容量に空きがない可能性があります。
- L3314 (F)Cannot delete temporary file
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。

14. 最適化リンケージエディタのエラーメッセージ

L3320 (F)Memory overflow

P3320 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。メモリを増やしてください。

L3400 (F)Cannot execute "ロードモジュール"

"ロードモジュール"を起動できません。"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。

L3410 (F)Interrupt by user

標準入力端末から「(cntl)+C」による割り込みを検出しました。

L3420 (F)Error occurred in "ロードモジュール"

"ロードモジュール"実行中にエラーが発生しました。

P3500 (F)Bad instantiation request file -- instantiation assigned to more than one file

インスタンス生成指定ファイルに誤りがあります。
リンク対象ファイルを再コンパイルしてください。

P3501 (F)Instantiation loop

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インスタンス生成要求ファイルと一致している可能性があります。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。

P3502 (F)Cannot create instantiation request file "ファイル"

インスタンス生成指定ファイルを作成できません。
オブジェクト作成ディレクトリ以下のアクセス権が正しいか確認してください。

P3503 (F)Cannot change to directory "ディレクトリ"

"ディレクトリ"に移動できません。
"ディレクトリ"が存在するか確認してください。

P3504 (F)File "ファイル" is read-only

"ファイル"は読み取り専用です。
アクセス権を変更してください。

L4000 (-)Internal error : ("内部エラー番号") "ファイル 行番号" / "コメント"

P4000 最適化リンケージエディタの処理中に内部的な問題が発生しました。
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、販売元のサポートセンタ までご連絡ください。

15. 標準ライブラリ構築ツール・ フォーマットコンバータのエラーメッセージ

15.1 エラー形式とエラーレベル

本章では、以下の形式で出力するエラーメッセージとエラー内容を説明します。

エラー番号 (エラーレベル) エラーメッセージ
 エラー内容

エラーレベルは、エラーの重要度に従い、3種類に分類されます。

	エラーレベル	動作
G1000 - G1999	(W) ウォーニング	処理を継続します。
G2000 - G2999	(E) エラー	オプション解析処理を継続し、処理を中断します。
G3000 - G3999	(F) フェータル	処理を中断します。

15.2 メッセージ一覧

G1001 (W) Debug information ignored

変換対象ファイルに #pragma option により、最適化あり指定の関数と最適化なし指定の関数が混在しました。デバッグ情報を削除して変換します。

G1002 (W) Command parameter specified twice

同じオプションを 2 回以上指定しています。同じオプションの中で、最後に指定したものを有効とします。オプションの指定に誤りが無いかどうか確認してください。

G2001 (E) Cannot open file "ファイル"

ファイルをオープンできません。ファイル名およびアクセス権が正しいか確認して下さい。

G2002 (E) Illegal file type "ファイル"

SYSROF から ELF への変換の場合、オブジェクトファイルまたはライブラリファイル以外のファイルが指定されました。ELF から SYSROF への変換の場合、ロードモジュールファイル以外のファイルが指定されました。ファイルの種別を確認の上、再実行して下さい。

G2003 (E) Illegal file format "ファイル"

ファイルのフォーマットが不正です。ファイルの内容を確認の上、再実行して下さい。

G3001 (F) Invalid command parameter "パラメータ"

不正なコマンドパラメータが指定されました。コマンドパラメータの内容を確認の上、再実行して下さい。

15. ライブラリ構築ツール・フォーマットコンバータのエラーメッセージ

- G3002 (F)No input file
入力ファイルがありません。
- G3003 (F)Command parameter buffer overflow
コマンドラインの指定が 32767 文字をこえています。
- G3101 (F)Cannot open file "ファイル"
ファイルをオープンできません。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。
- G3102 (F)Cannot input file "ファイル"
指定されたファイルから入力できません。変換対象ファイルをアクセスしていないか確認してください。
- G3103 (F)Cannot create file "ファイル"
ファイルを生成できません。ディスク容量に空きがあるか確認してください。
- G3104 (F)Cannot output file "ファイル"
ファイルに書き込めません。書き込み禁止を解除してください。
- G3105 (F)Cannot open internal file
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。中間ファイルをアクセスしていないか確認してください。
- G3106 (F)Cannot output internal file
内部で生成する中間ファイルに出力できません。ディスク容量に空きがないか、ディスクに物理的なエラーがある場合があります。
- G3107 (F)Memory overflow
内部で使用するメモリ領域を割り当てることができません。必要なメモリを確保して再実行してください。
- G3108 (F)Illegal format in archive "ファイル"
指定されたファイルはアーカイブのフォーマットではありません。
- G3201 (F)Cannot execute compiler
コンパイラを起動できません。コンパイラのパス名を確認してください。
- G3202 (F)Cannot execute optlinker
最適化リンケージエディタを起動できません。最適化リンケージエディタのパス名を確認してください。
- G3203 (F) Interrupt by user
実行中に割り込みを検出しました。
- G3300 (F)Already existent file "ファイル"
ファイルは既に存在しています。

16. 限界値

16.1 コンパイラの限界値

コンパイラの限界値を表 16.1 に示します。

ソースプログラムを作成する際は、この限界値の範囲で作成してください。

表 16.1 コンパイラの限界値

分類	項目	限界値
1	起動	define オプションが指定可能なマクロ名総数
2		ファイル名長
3	ソース	1 行の文字数
4	プログラム	1 ファイルあたりのソースプログラムの行数
5		コンパイル可能なソースプログラムの総行数
6	プリプロセッサ	#include 文のネストの深さ
7		#define 文のマクロ名総数
8		マクロ定義、マクロ呼び出しの指定可能引数
9		マクロ名の再置き換えの数
10		#if, #ifdef, #ifndef, #else, #elif 文のネストの深さ
11		#if, #elif 文で指定可能な演算子、非演算子の合計数
12	宣言	関数定義数
13		外部結合となる識別子(外部名)の数
14		1 関数内で有効な識別子(内部名)の数
15		基本型を修飾するポインタ型、配列型、関数型の合計数
16		配列の次元数
17		配列・構造体のサイズ
18	文	複文のネストの深さ
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
20		1 関数内で指定可能な goto ラベルの数
21		switch 文の数
22		switch 文のネストの深さ
23		1 つの switch 文内で指定可能な case ラベルの数
24		for 文のネストの深さ
25	式	文字列の長さ
26		関数定義、関数呼び出しで指定可能引数
27		1 つの式で指定可能な演算子と非演算子の合計数
28	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

【注】*1 非静的関数メンバの場合は 62 個になります。

*2 詳細は「9.2.2(4) C/C++ライブラリ関数の初期設定(_INITLIB)」を参照してください。

16.2 アセンブラの限界値

アセンブラの限界値を表 16.2 に示します。

表 16.2 アセンブラの限界値

項目	限界値
1 1行文字数	8192 文字
2 文字定数	4 文字まで
3 シンボル長	制限なし ^{*1}
4 シンボル数	制限なし
5 外部参照シンボル数	制限なし
6 外部定義シンボル数	制限なし
7 セクションの最大サイズ	H'FFFFFFFF バイトまで
8 セクション数	<i>H'FEF1 個(デバッグあり)、H'FEFA 個(デバッグなし)</i>
9 ファイルインクルード	ネストは 30 レベルまで
10 文字列長	255 文字まで
11 ファイル名長	制限なし(OS に依存)

【注】 本バージョンで制限値が変更となった項目は、太字 斜体で示しています。

*1 プリプロセッサ変数名、DEFINE 置換シンボル名、マクロ名および、マクロ仮引数名は 32 文字までです。

17. バージョンアップにおける注意事項

17.1 バージョンアップ時の注意事項

旧バージョン(「SuperH RISC engine C/C++コンパイラパッケージ」Ver.6.x 以前)からバージョンアップして使用する場合の注意事項を説明します。

17.1.1 プログラムの動作保証

コンパイラをバージョンアップしてプログラム開発する場合、プログラムの動作が変わることがあります。プログラムを作成する際は、以下の点に注意して、お客様のプログラムを十分にテストしてください。

(1) プログラムの実行時間やタイミングに依存するプログラム

言語仕様は、プログラムの実行時間については何も規定していません。したがってコンパイラのバージョンの違いによりプログラムの実行時間と I/O 等周辺機器のタイミングのずれ、あるいは割り込み処理のような非同期処理の時間の差等により、プログラムの動作が変わる場合があります。

(2) 1つの式に2個以上の副作用が含まれているプログラム

1つの式に2個以上の副作用が含まれている場合、コンパイラのバージョンにより、動作が変わる可能性があります。

例：

```
a[i++] = b[i++];          /* i のインクリメント順序は不定です。          */
f(i++, i++) ;           /* インクリメントの順序でパラメタの値が変わります。 */
                        /* i の値が 3 の時 f(3, 4) または f(4, 3) になります。 */
```

(3) 結果がオーバーフローや不当演算に依存するプログラム

オーバーフローが生じた場合や、不当演算を実施した場合、結果の値は保証しません。したがって、バージョンが変わると動作が変わる可能性があります。

例：

```
int a, b;
x = (a*b)/10; /* a と b の値の範囲によってはオーバーフローする可能性があります */
```

(4) 変数の初期化抜け、型の不一致

変数が初期化されていない場合や、パラメタやリターン値の型が呼び出し側と呼び出される側で対応していない場合、不正な値をアクセスすることになります。したがって、コンパイラのバージョンによって動作が変わる場合があります。

17. バージョンアップにおける注意事項

例：

<pre>file 1: int f(double d) { : }</pre>	<pre>file 2: int g(void) { f(1); }</pre>	<p>関数呼び出し側のパラメータは int型ですが、関数定義側のパラメータは、double型のため、値を正しく参照できません。</p>
--	--	---

上記に記載された情報が全ての起こりうる状況を示したわけではありません。したがって、お客様の責任で本コンパイラを正しくご使用の上、お客様のプログラムを十分にテストしてください。

17.1.2 旧バージョンとの互換性

旧バージョンのコンパイラ(Ver.5.x 以前)、アセンブラ(Ver.4.x 以前)およびリンケージエディタ(Ver.6.x 以前)出力のオブジェクトファイル、ライブラリファイルとリンクする場合、または旧バージョンで使用していたデバッグをそのまま使用する場合に注意すべき点を説明します。

(1) オブジェクト形式

オブジェクトファイル形式は、従来の SYSROF から標準フォーマットの ELF 形式に変更しました。また、デバッグ情報形式も、標準フォーマットの DWARF2 形式に変更しました。

旧バージョンのコンパイラ(Ver.5.x 以前)、アセンブラ(Ver.4.x 以前)出力オブジェクトファイル(SYSROF)を最適化リンケージエディタに入力する場合は、ファイルコンバータを使用して ELF 形式に変換してください。ただし、リンケージエディタ出力リロケータブルファイル(拡張子 rel)およびリロケータブルファイルを含むライブラリファイルは変換できません。

また、SYSROF 形式および ELF/DWARF1 形式ロードモジュールをサポートするデバッグを使用する場合は、ファイルコンバータを使用して ELF/DWARF2 形式ロードモジュールを、SYSROF または ELF/DWARF1 形式に変換してください。

(2) インクルードファイルの基点

ディレクトリ相対形式で指定されたインクルードファイル検索時、旧バージョンではコンパイラ起動ディレクトリを基点に検索していましたが、ソースファイルのあるディレクトリを基点に検索するように変更しました。

(3) C++プログラム

エンコード規則、実行方式を変更しましたので、旧バージョンのコンパイラで作成した C++ オブジェクトファイルはリンクできません。必ずリコンパイルしてから使用してください。

また実行環境の設定で用いる、グローバルクラスオブジェクト初期処理 / 後処理のライブラリ関数名も変更になりました。「9.2.2 実行環境の設定」を参照し、修正してください。

(4) コモンセクションの廃止(アセンブリプログラム)

オブジェクトフォーマットの変更に伴い、コモンセクションのサポートを廃止しました。

(5) .END のエントリ指定(アセンブリプログラム)

.END でエントリ指定できるシンボルは外部定義シンボルだけになりました。

(6) モジュール間最適化

旧バージョンのコンパイラ(Ver.5.x 以前)、アセンブラ(Ver.4.x 以前)出力オブジェクトファイルは、モジュール間最適化の対象になりません。モジュール間最適化の対象にしたいファイルについては、必ずリコンパイル、リアセンブルしてください。

17.1.3 コマンドラインインタフェース

(1) アセンブラ、最適化リンケージエディタコマンドライン規約

ファイル名、オプション間に、スペースが必須になりました。
また、ファイル名、オプションの指定順序に制限がなくなりました。

(2) 最適化リンケージエディタオプション

会話形式のオプション指定サポートを廃止しました。

また、旧バージョンのモジュール間最適化ツール(optlnksh)とリンケージエディタ(lnk)、ライブラリアン(lbr)、オブジェクトコンバータ(cnvs)を統合しました。これに伴い、コマンドライン仕様が大幅に変更になりました。変更したコマンド一覧を表 17.1、表 17.2 に示します。

表 17.1 リンケージコマンド変更一覧

No.	コマンド名	V6	V7	備考
1	start	start= セクション(アドレス) 短縮形 st	start= セクション/アドレス 短縮形 star	
2	rom	rom=(rom セクション, ram セクション)	rom=rom セクション= ram セクション	
3	define	define=外部名(定義値)	define=外部名=定義値	
4	rename	rename= ed=変更前(変更後), er=変更前(変更後), un=変更前(変更後) 短縮形 re	rename= (変更前=変更後), (変更前=変更後), (変更前=変更後), 短縮形 ren	オブジェクト形式 変更によりユニッ トの概念廃止
5	delete	delete= ed=ユニット.シンボル un=ユニット	delete=(シンボル)	オブジェクト形式 変更によりユニッ トの概念廃止
6	print / noprint	print noprint	list	ファイル名省略可
7	mlist	mlist	list	
8	information	information	message	
9	directory	directory	HLNK_DIR(環境変数)	
10	form	短縮形 f	短縮形 fo	
11	output / nooutput	短縮形 o nooutput 指定可	短縮形 ou nooutput 指定不可	output のみ指定可
12	cpu	短縮形 c	短縮形 cp	直接範囲指定可
13	elf / sysrof / sysrofplus	elf / sysrof / sysrofplus	廃止	常に ELF
14	exclude / noexclude	exclude / noexclude	廃止	常に exclude
15	align_section	align_section	廃止	常に有効

17. バージョンアップにおける注意事項

No.	コマンド名	V6	V7	備考
16	check_section	check_section	廃止	常に有効
17	cpucheck	cpucheck	廃止	常に有効
18	udf / noudf	udf / noudf	廃止	常に出力
19	udfcheck	udfcheck	廃止	常に有効
20	echo / noecho	echo / noecho	廃止	常に抑止
21	exchange	exchange	廃止	オブジェクト形式 変更によりユニット の概念廃止
22	autopage	autopage	廃止	対象 cpu なし
23	abort	abort	廃止	会話形式廃止
24	list	list	廃止	V7 の list オプションとは別
25	library / nolibrary	nolibrary 指定可	nolibrary 指定不可	library のみ指定可
26	exit	省略不可	省略可	
27	debug / nodebug	省略時 : nodebug	省略時 : 入力ファイルの debug 情報有無に依存	

【注】* change_message オプションで無効にすることができます。

表 17.2 ライブラリアンコマンド変更一覧

No.	コマンド名	V2	V7	備考
1	add	add	input	
2	directory	directory	HLNK_DIR(環境変数)	
3	slist	slist	list show	
4	list	list (s)	list show	
5	delete	短縮形 d	短縮形 del	
6	create	create (s u)	output form=library(s u)	
7	output	output (s u) 短縮形 o	output form=library(s u) 短縮形 ou	
8	replace	短縮形 r	短縮形 rep	
9	abort	abort	廃止	会話形式廃止
10	exit	省略不可	省略可	

17.1.4 提供内容

「SuperH RISC engine C/C++コンパイラパッケージ」の提供内容のうち、以下のファイルが変更になりました。

(1) 標準ライブラリファイル

関数インタフェースや最適化オプションを任意に指定できるようにするため、従来の標準ライブラリファイル提供から、標準ライブラリ構築ツール提供に変更しました。

(2) ヘッダファイル

標準ライブラリとして提供する初期設定用ルーチン `_INITSCT` 関数、`_CALL_INIT` 関数、`_CALL_END` 関数用のヘッダファイル `_h_c_lib.h` を追加しました。

17.1.5 リストファイル仕様

(1) 最適化リンケージエディタ

従来のリンケージマップリスト、ライブラリリストのフォーマットを一新しました。

17.2 追加・改善内容

17.2.1 共通の追加・改善(パッケージ Ver.6)

(1) 制限値の緩和

ソースプログラムやコマンドラインの制限を大幅に緩和しました。

- ファイル名長：251 バイト 無制限
- シンボル長：251 バイト 無制限
- シンボル数：32,767 個 無制限
- ソースプログラム行数：C/C++:32767 行、ASM:65535 行 無制限
- C プログラム文字列長：512 文字 32766 文字
- アセンブリプログラム行長：255 文字 8192 文字
- サブコマンドファイル行長：ASM:300 バイト、optlink:512 バイト 無制限
- 最適化リンケージエディタ rom オプションのパラメタ数:64 個 無制限

(2) ディレクトリ名、ファイル名のハイフン(-)

ディレクトリ名、ファイル名にハイフン(-)を指定できるようになりました。

(3) コピーライト表示抑止

logo/nologo オプション指定により、コピーライト表示有無を指定できるようになりました。

(4) エラーメッセージのプリフィックス

日立統合開発環境でのエラーヘルプ機能サポートに伴い、コンパイラ、最適化リンケージエディタのエラーメッセージの先頭にプリフィックスを付与しました。

17.2.2 コンパイラの追加・改善機能

(1) Ver.6 の主な追加・改善機能

(a) fpscr オプションの追加

cpu=sh4 オプションを指定しかつ fpu オプションを指定していないとき、関数呼び出し前後の FPSCR レジスタの精度モードを保証するかどうかを指定できるようになりました。

(b) #pragma 拡張子

#pragma 拡張子が()なしで記述できるようになりました。

(c) 組み込み関数の追加

trace 関数を追加しました。

(d) 暗黙の宣言の追加

__HITACHI__、__HITACHI_VERSION__が暗黙に#define 宣言されます。

(e) static ラベル名

#pragma inline_asm でファイル内 static ラベルを参照できるように、ラベル名を__\$(名前)に変更しました。ただし、リンケージリストでは_(名前)と表示されます。

(f) 言語仕様拡張・変更

- 共用体初期化時のエラーを抑止します。

例：

```
union{
    char c[4];
} uu={{'a','b','c'}};
```

- 構造体の代入と宣言を同時にできるようになりました。

例：

```
struct{
    int a, int b;
} s1;

void test()
{
    struct S s2=s1;
}
```

- bool 型データの境界調整数が 4byte になりました。
- C++言語仕様として、例外処理やテンプレート機能もサポートしました。
- C プリプロセッサが ANSI/ISO 対応しました。

(2) Ver.7 の主な追加・改善機能

(a) 外部変数アクセス最適化機能(map オプションサポート)

リンク時の変数、関数の割り付けアドレスに基づいた外部変数アクセス、関数分岐命令の最適化を行います。一度目のリンク時に最適化リンケージエディタより出力(map オプション指定)された外部シンボル割り付け情報ファイルを用いてリコンパイルすることにより、最適化を実施します。

(b) GBR 相対アクセスコード自動生成(gbr オプションサポート)

gbr=auto を指定した場合、コンパイラが GBR の設定および GBR 相対アクセスコードを自動生成します。関数呼び出し前後で GBR の値を保証します。ただし、GBR 関連の組み込み関数は使用できません。

(c) speed/size 選択オプションの強化

speed/size 選択オプション(shift、blockcopy、division、approxdiv オプション)を追加し、より細かい size/speed 調整を可能にしました。

(d) 組み込み向け機能の強化

- 組み込み関数の追加
倍精度乗算、SWAP 命令、LDTLB 命令、NOP 命令
- #pragma 拡張子の追加、変更
 - #pragma entry エントリ関数指定、SP の設定
 - #pragma stacksize スタックサイズの指定
 - #pragma interrupt sp=<変数>+<定数>、sp=&<変数>+<定数>のサポート
- セクション演算子のサポート
セクションのアドレス、サイズ参照を C 言語で記述できる機能をサポートしました。
- アドレスキャストのエラー緩和
外部変数初期化時のアドレス初期化に対するキャスト式のエラーを緩和しました。

(e) ライブラリの改善

- リエントラントライブラリサポート
ライブラリ構築ツールで reent オプションを指定した場合、リエントラントライブラリが生成されます。
- malloc 確保サイズ単位、入出力ファイル数の可変性
C/C++ライブラリ関数の初期設定において、_sbrk_size で malloc 確保サイズを、_nfiles で入出力ファイル数を指定できるようになりました。これにより、RAM 容量を節約できます。指定を省略した場合、malloc 確保サイズは 520、入出力ファイル数は 20 になります。
- 簡易 I/O のサポート
ライブラリ構築ツールで nofloat オプションを指定した場合、浮動小数点変換をサポートしない、サイズの小さい I/O ルーチンを生成します。

17.2.3 最適化リンケージエディタの追加・改善機能

(1) Ver.7 の主な追加・改善機能

(a) ワイルドカードのサポート

入力ファイルや start オプションのセクション名でワイルドカードを指定できます。

(b) サーチパス

環境変数(HLNK_DIR)により、複数の入力ファイル、ライブラリファイルのサーチパスを指定できます。

(c) ロードモジュール分割出力

アブソリュートロードモジュールファイルを分割出力できます。

(d) エラーレベルの変更

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は、個別にエラーレベルや出力有無を変更できます。

(e) バイナリ、HEX サポート

バイナリファイルを入出力できるようになりました。
また、インテル HEX タイプの出力も選択できるようになりました。

(f) stack 使用量情報の出力

stack オプションにより、スタック解析ツール用情報ファイルを出力できます。

(g) デバッグ情報削除機能

strip オプションにより、ロードモジュールファイルやライブラリファイル内のデバッグ情報だけを削除できます。

(2) Ver.7.1 の主な追加・改善機能

(a) 外部シンボル割り付け情報ファイルの出力(map オプションサポート)

map オプションを指定した場合、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生成します。

17.3 フォーマットコンバータ操作方法

17.3.1 オブジェクトファイル形式

オブジェクトファイル形式は、標準フォーマットの ELF 形式に準拠しています。また、デバッグ情報形式も、標準フォーマットの DWARF2 形式に準拠しています。

17.3.2 旧バージョンとの互換性

(1) オブジェクトファイル、ライブラリファイル

旧バージョンのコンパイラ(Ver.5.x 以前)、アセンブラ(Ver.4.x 以前)出力オブジェクトファイルおよびライブラリファイルを最適化リンケージエディタに入力する場合は、フォーマットコンバータを使用して ELF 形式に変換してください。ただし、デバッグ情報は変換時に削除されます。

また、リンケージエディタ出力リロケータブルファイル(拡張子 rel)およびリロケータブルファイルを含むライブラリファイルは変換できません。

フォーマットコンバータは、オブジェクト形式を変換したファイルを、入力ファイル名と同じ名前前で出力します。入力ファイルは、<入力ファイル名.拡張子>.bak として保存します。

ELF 形式のオブジェクトファイルおよびライブラリファイルを、旧バージョンのコンパイラ(Ver.5.x 以前)、アセンブラ出力オブジェクト形式に変換することはできません。

(2) ロードモジュールファイル

ELF 形式のロードモジュールファイルは、フォーマットコンバータを使用して旧バージョンのリンケージエディタ(Ver.6.0 以前)出力のフォーマットに変換することができます。変換可能なオブジェクトファイル形式を、表 17.3 に示します。

表 17.3 ELF 形式から変換可能なオブジェクトファイル形式

1	コンパイラ バージョン アセンブラ	リンケージエディタ 指定オプション	オブジェクトファイル形式		変換可否	
			オブジェクト	デバッグ情報		
1	Ver.4.x 以前	debug	SYSROF	SYSROF		
2	Ver.3.x	sdebug	SYSROF	SYSROF	x	
3		sysrof	debug	SYSROF	SYSROF	
4	Ver.5.x		sdebug	SYSROF	DWARF1	x
5	Ver.4.x	elf	debug	ELF	DWARF1	
6			sdebug	ELF	DWARF1	x

フォーマットコンバータは、オブジェクト形式を変換したファイルを、入力ファイル名と同じ名前前で出力します。入力ファイルは、<入力ファイル名.拡張子>.bak として保存します。

旧バージョンのリンケージエディタ(Ver.6.0 以前)出力ロードモジュールファイルを、ELF 形式に変換することはできません。

本バージョンのコンパイラ、アセンブラ、最適化リンケージエディタで新規追加した機能を使用した場合は、変換できません。

17.3.3 オプション指定規則

コマンドラインの形式は以下の通りです。

```
helcncv[ <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>]
<ファイル名> : ワイルドカード(*,?)も指定できます。
```

17.3.4 オプション解説

コマンドライン形式の場合は、英大文字は短縮形指定時の文字を示します。下線は省略時解釈を示します。日立統合開発環境を使用する場合は、最適化リンケージエディタのオプションウィンドウで指定します。対応するダイアログメニューを、タブ名[項目]で示します。

変換対象ファイルの種別(オブジェクトファイル、ライブラリファイル、ロードモジュール)は、フォーマットコンバータが自動判定します。

(1) オブジェクトファイル、ライブラリファイルの変換

旧バージョンのコンパイラ(Ver.5.x 以前)、アセンブラ(Ver.4.x 以前)で作成したりロケータブルオブジェクトファイル、ライブラリファイルを、ELF 形式に変換します。オブジェクトファイル、ライブラリファイル内に含まれているデバッグ情報は削除されます。

本機能は、日立統合開発環境のオプションではサポートしていません。コマンドラインで使用してください。

表 17.4 オブジェクトファイル、ライブラリファイル変換用オプション一覧

項目	オプション	ダイアログメニュー	指定内容
1	アドレス空間の指定 ^{*1}	Address_space=<サイズ> <サイズ>:20 24 28 32	アドレス空間の指定
2	fpu	Fpu	FPU あり
3	dsp	Dsp	DSP あり
4	エンディアンの指定	Endian = { Big Little }	エンディアンの指定

【注】*1 H8S,H8/300 シリーズ用のオプションです。SuperH では無効です。

FPU あり指定***Fpu***

- 書 式 Fpu
- 説 明 CPU が SH-2E、SH-4 の場合に指定します。
- 例 helfcnv -fpu *.obj *.lib ; ディレクトリ内全ての*.obj,*.lib を elf 形式に変換

DSP あり指定***Dsp***

- 書 式 Dsp
- 説 明 CPU が SH2-DSP、SH3-DSP の場合に指定します。
- 例 helfcnv -dsp *.obj ; ディレクトリ内全ての*.obj を elf 形式に変換

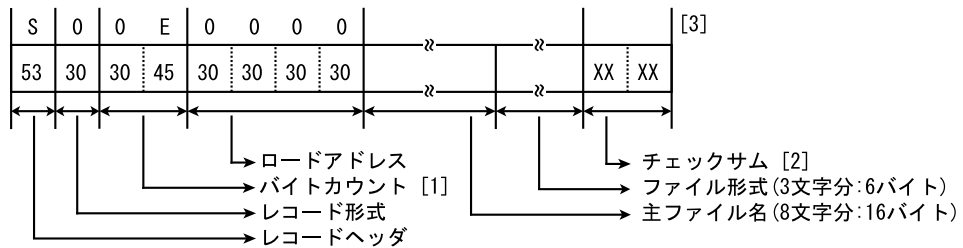
18. 付録

18.1 Sタイプ、HEXファイル形式

本節では、最適化リンカージエディタによって出力されるSタイプファイルおよび、HEXファイルについて説明します。

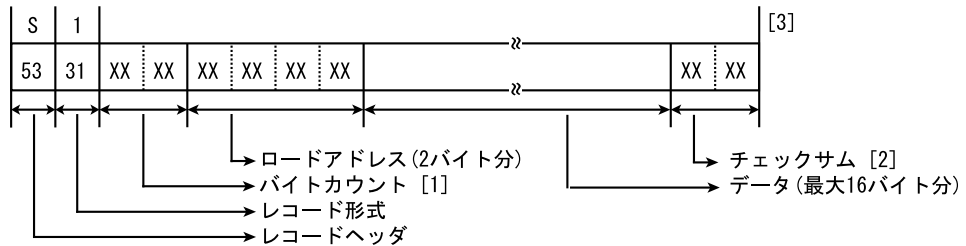
18.1.1 Sタイプファイル形式

(a) ヘッダレコード(S0レコード)

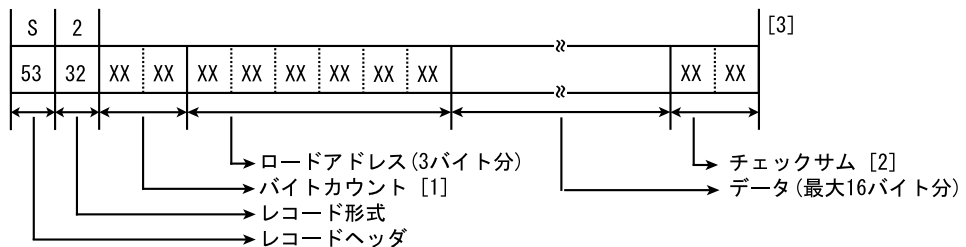


(b) データレコード(S1, S2, S3レコード)

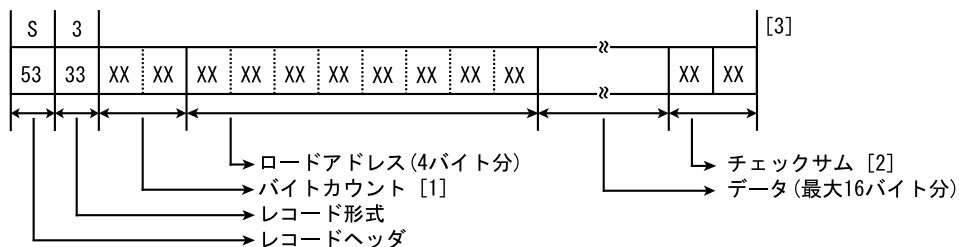
(i) ロードアドレスが0 ~ FFFFの場合



(ii) ロードアドレスが10000 ~ FFFFFFFの場合

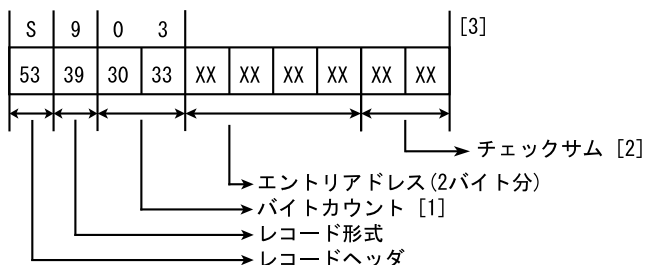


(iii) ロードアドレスが1000000 ~ FFFFFFFFの場合

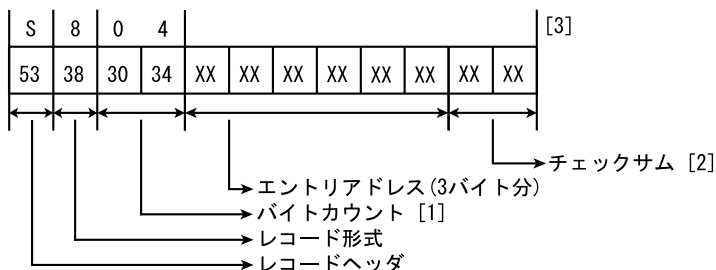


(c) エンドレコード (S9, S8, S7レコード)

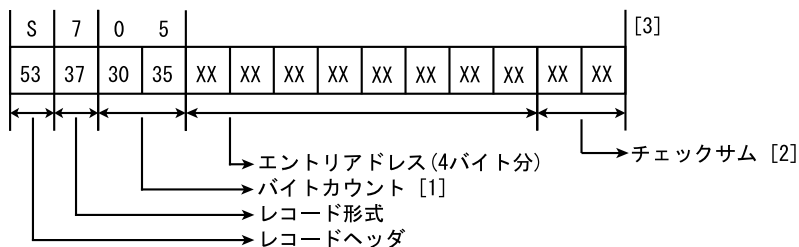
(i) エントリアドレスが0 ~ FFFFの場合



(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



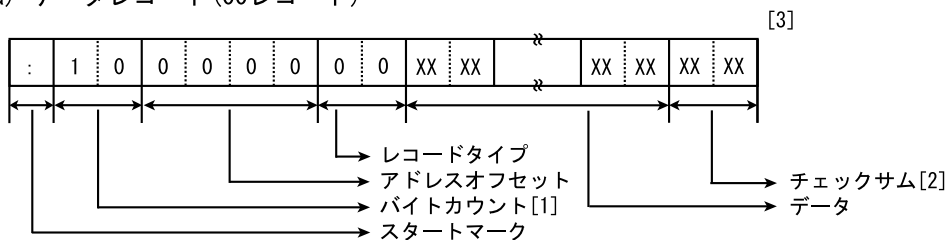
- 【注】
- [1] ロードアドレス(またはエントリアドレス)からチェックサムまでのバイト数
 - [2] バイトカウンタからチェックサムの前までのデータ値をバイト単位に加算した結果の1の補数
 - [3] チェックサムの直後に改行コードが付加される

18.1.2 HEX ファイル形式

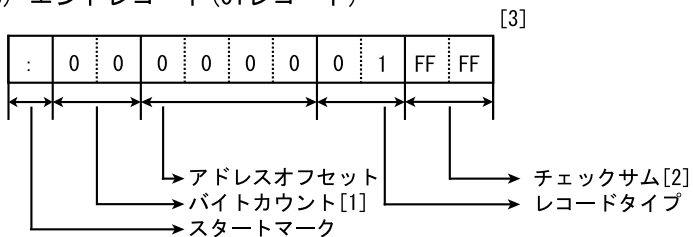
各データレコードの実行アドレスは以下のように求めます。

- (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 $\ll 4$) + (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 $\ll 16$) + (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)



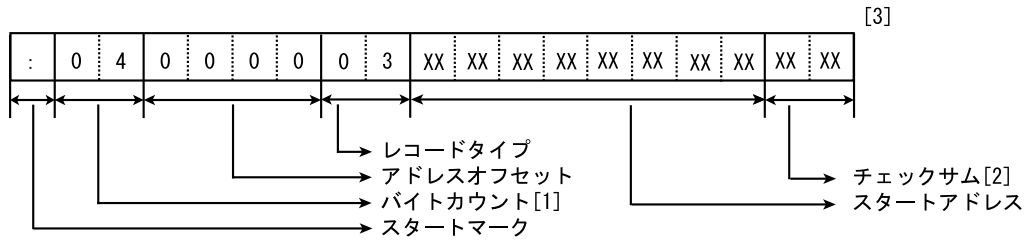
(b) エンドレコード (01レコード)



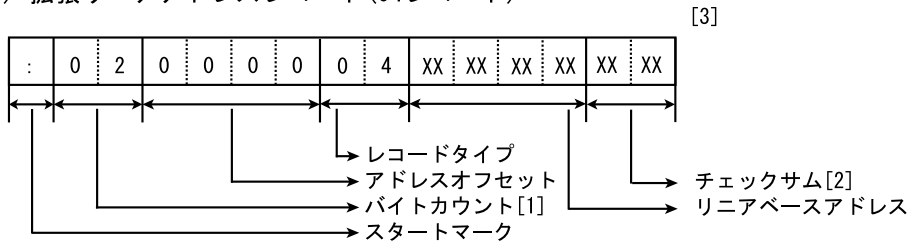
(c) 拡張セグメントアドレスレコード (02レコード)



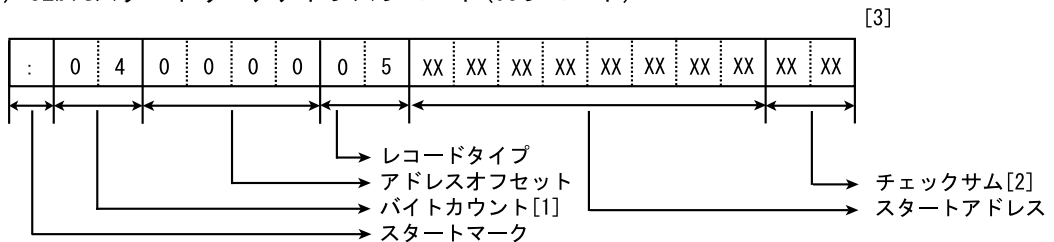
(d) スタートアドレスレコード(03レコード)



(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] レコードタイプからチェックサムの前までのバイト数
 [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

18.2 ASCII コード一覧表

表 18.1 ASCII コード一覧表

下位 4 ビット	上位 4 ビット							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

索引

英語

記号・数字

#pragma abs16	205
#pragma entry	213
#pragma gbr_base	215
#pragma gbr_base1	215
#pragma global_register	214
#pragma inline	209
#pragma inline_asm	210
#pragma interrupt	206
#pragma noregalloc	212
#pragma noregsave	212
#pragma regsave	212
#pragma section	204
#pragma stacksize	205
#pragma 拡張子	203
\$EVEN	499
\$EVEN2	499
\$G0	122
\$G1	122
\$ODD	499
\$ODD2	499
*指定	313, 314
.AELIF	588
.ELSE	588, 590
.AENDI	588, 590
.AENDR	591
.AENDW	592
.AERROR	594
.AIF	588
.AIFDEF	590
.ALIGN	538
.ALIMIT	594
.AREPEAT	591
.ASSIGN	540
.ASSIGNA	584
.ASSIGNC	586

.AWHILE.....	592
.CPU	533
.DATA	543
.DATAB.....	544
.DEBUG.....	562
.DEFINE.....	587
.END	576
.ENDIAN.....	563
.ENDM	598
.EQU	539
.EXITM.....	593
.EXPORT.....	557
.FDATA.....	549
.FDATAB	550
.FORM.....	570
.FREG	542
.FRES.....	556
.GLOBAL	559
.HEADING	571
.IMPORT	558
.INCLUDE.....	578
.INSTR.....	606
.LEN	605
.LINE	565
.LIST.....	567
.MACRO.....	598
.ORG.....	537
.OUTPUT.....	560
.PAGE.....	572
.PRINT.....	566
.PROGRAM.....	574
.RADIX.....	575
.REG	541
.RES.....	552
.SDATA.....	545
.SDATAB	546
.SDATAC	547
.SDATAZ.....	548
.SECTION	534
.SPACE.....	573
.SRES.....	553
.SRESC.....	554
.SRESZ.....	555
.SUBSTR.....	607

.XDATA	551
_ _DATE_	185
_ec2p_os	389
_ _FLT_	100
_ _HITACHI_	100
_ _HITACHI_VERSION_	100
_ _secend	216
_ _sectop	216
_ _TIME_	185
_B_cnt_ptr	374
_B_len_ptr	374
_BIG	100
_CALL_END	137
_CALL_INIT	137
_CLOSEALL	137, 147
_DON	100
_ec2p_new_handler	398
_file_Ptr	396
_FLT	100
_fmtmask	365
_FPD	100
_FPS	100
_h_c_lib.h	727
_im	400, 408
_Init	396
_INIT_IOLIB	145
_INIT_LOWLEVEL	145
_INIT_OTHERLIB	145
_INITLIB	137, 145
_INITSCT	137
_IOFBF	300
_IOLBF	300
_IONBF	300
_LIT	100
_PIC	100
_re	400, 408
_RON	100
_SH1	100
_SH2	100
_SH2E	100
_SH3	100
_SH4	100
_statemask	365
~mystrbuf	396

10 進数字	254
10 進数字判定	257
16 進数字	254
16 進数字判定	260
1 文字出力	328
1 文字入力	326
2 バイトアドレス指定	205
4×4 行列の乗算	243
4×4 行列の乗算と差	244
4×4 行列の乗算と和	244
4 バイトデータ並べ替え	236
4 次元ベクタの 4×4 行列による変換と 4 次元ベクタとの差	241
4 次元ベクタの 4×4 行列による変換と 4 次元ベクタとの和	240
4 次元ベクタの差	242
4 次元ベクタの和	242
64bit 乗算	233, 234

A

abort	56, 163, 433
abs	101, 345, 406, 413
abs16	12
ABS16 宣言	12
absolute	67
absolute_forbid	77
acos	269
acosf	283
ADD	508
add4	242
ADDC	508
ADDV	508
adjustfield	365
aggressive	28
ALIGN	534
align16	25
all	12, 90
AND	509
app	365
aproxdiv	27
arg	406, 413
ASCII コード	741
asctime	433
asin	269
asinf	283
asmcode	9

assert	253
assert.h	253
assigna	41
assignc	42
atan	270
atan2	271
atan2f	285
atanf	284
ate	365
atexit	162, 433
atof	337
atoi	337
atol	338
auto	12, 20
auto_literal	52
B	
B	121, 503, 543, 544, 552, 575
B'	488
B_beg_pptr	374
B_beg_ptr	374
B_end_ptr	374
B_next_pptr	374
B_next_ptr	374
badbit	365
basefield	365
beg	366
BF	510
BF/S	511
big	30, 54, 734
BIG	563
Big Endian	194
bin	101
binary	64, 67, 365
blockcopy	22
bls	101
bool	186
boolalpha	365, 371
BRA	510
BRAF	511
branch	44, 75
bsearch	344
BSR	510
BSRF	511

bss	10
BT	510
BT/S	511
BTBL	136, 145
BUFSIZ	300
C	
c	35, 101
C	121
C\$INIT	122
C\$VTBL	122
C/C++ライブラリの初期設定	137
C/C++ライブラリ関数の初期設定	145
C/C++言語選択	35
C_flg_ptr	374
C++仮想関数表領域	122
C++初期処理 / 後処理データ領域	122
cache size	77
cal	101
CALL	567
calloc	342
calls	49
case	21
cc	101
ceil	279
ceilf	293
change_message	86
char	186
CHAR_BIT	265
CHAR_MAX	265
CHAR_MIN	265
chcount	381
clearerr	334
clock	433
close	151, 396
CLRMAC	510
CLRS	514
CLRT	510
CMP/EQ	508
CMP/GE	508
CMP/GT	508
CMP/HI	508
CMP/HS	508
CMP/PL	508

CMP/PZ	508
CMP/STR	508
code	9, 49, 121
CODE	534, 567
COL	570
columns	59
comment	24
complex	90, 400
compress	83
COND	567
conditionals	49
conj	406, 414
const	10, 11
const_iterator	416
ConvComplete	466
ConvCyclic	467
ConvPartial	468
CopyFromX	474
CopyFromY	474
CopyToX	473
CopyToY	473
CopyXtoY	472
CopyYtoX	472
CorrCyclic	470
Correlate	469
cos	272, 406, 414
cosf	286
cosh	273, 406, 414
coshf	287
cp	101
cpp	35, 101
cppstring	90
cpu	13, 30, 53, 80
CPU 種別	30, 53, 533
CREF	566
cross_reference	50
ctime	433
ctype	90
ctype.h	254
cur	366
D	
D	121, 503, 549, 550, 556, 575
D'	488

data.....	10, 11, 121
DATA.....	534
dbg.....	101
DBG.....	560
DBL_DIG.....	263
DBL_EXP_DIG.....	263
DBL_MANT_DIG.....	263
DBL_MAX.....	262
DBL_MAX_10_EXP.....	262
DBL_MAX_EXP.....	262
DBL_MIN.....	262
DBL_MIN_10_EXP.....	263
DBL_MIN_EXP.....	262
DBL_NEG_EPS.....	263
DBL_NEG_EPS_EXP.....	264
DBL_POS_EPS.....	263
DBL_POS_EPS_EXP.....	263
DCF.....	524
DCT.....	524
debug.....	10, 43, 69
dec.....	365, 373
DEF.....	567
define.....	7, 40, 64
definitions.....	49
delete.....	84
denormalize.....	32, 55
difftime.....	433
Dlir.....	458
Dlir1.....	459
div.....	346
div_t.....	336
DIV0S.....	508
DIV0U.....	508
DIV1.....	508
division.....	13
DMULS.....	511
dmuls_h.....	233
dmuls_l.....	233
DMULU.....	511
dmulu_h.....	234
dmulu_l.....	234
double.....	31, 186
double = float.....	33
double_complex.....	

– double_complex	408
– double_complex	408
– real	408
– imag	408
– operator=	409
– operator+=	409
– operator-=	409
– operator*=	409
– operator/=	409
– operator=	409
– operator+=	409
– operator-=	409
– operator*=	409
– operator/=	409
double_complex クラス	408
double_complex メンバ外関数	410
double float 変換	33
double 型	198
dsp	733
DSP あり指定	733
DSP シフト演算命令	526
DSP ライブラリ	434
DSP レジスタ直接	526, 529
DSP 演算オペレーション	523, 524
DSP 演算命令	526
DSP 演算命令一覧	528
DSP 算術演算命令	526
DSP 命令	523, 526
DSP 論理演算命令	526
DT	511
DTBL	136, 145
DUMMY	534
dwarf1	735
E	
EBADF	266, 686
EC++クラスライブラリ	361
ECBASE	266, 685
ecpp	24
ECSPEC	266, 686
EDBLO	266, 685
EDBLU	266, 685
EDIV	266, 685
EDOM	266, 267, 281, 685
EDSP_BAD_ARG	434

EDSP_NO_HEAP	434
EDSP_OK.....	434
EEXP	266, 685
EEXPN	266, 685
EFFTALLSCALE.....	438
EFFTMIDSCALE.....	438
EFFTNOSCALE.....	438
EFLOATO	266, 685
EFLOATU	266, 685
EIOBFRSM	686
EIOBRESM	686
ELDBLO.....	266, 685
ELDBLU.....	266, 685
EMALFRSM	686
EMALRESM	686
end	87, 366
end_cnvl.....	236
endian.....	30, 54, 734
endl	393
ends	393
ensigdsp.h	434
entry	65
enum	186
eof	374
EOF.....	248, 300
eofbit.....	365
EQ.....	588, 592
ERANGE	266, 267, 281, 685
errno.....	252, 266
errno.h.....	266
errno_addr.....	153
errno アドレス取得.....	153
error	56, 86
ESTRN.....	266, 685
ETLN	266, 685
ETOKFRSM.....	686
ETOKRESM.....	686
euc.....	36, 37, 58
EUC	58
exception.....	33
exit	87, 162, 433
exp	101, 275, 407, 414
EXP.....	567
expand.....	44

expansion.....	15
expansions.....	49
expf.....	289
extract.....	85
EXTS.....	508
EXTU.....	508
fabs.....	279
FABS.....	512, 516
fabsf.....	293
FADD.....	512, 516
failbit.....	365
fclose.....	304
FCMP/EQ.....	512, 516
FCMP/GT.....	512, 516
FCNVDS.....	517
FCNVSD.....	517
FDIV.....	512, 516
feof.....	334
ferror.....	335
fflush.....	304
FftComplex.....	439
FftInComplex.....	443
FftInReal.....	444
FftReal.....	440
FFT 構造.....	438
fgetc.....	322
fgetpos.....	433
fgets.....	323
FILE.....	300
FILE 型.....	147
FILE 構造体.....	248
fillch.....	364
filt_ws.h.....	434
fipr.....	238
FIPR.....	516
FIPR 命令.....	238
Fir.....	454
FIR.....	454
Fir1.....	455
FIR 作業領域解放.....	464
FIR 作業領域割り付け.....	462
fixed.....	365, 373
FLDI0.....	512, 515

FLDI1	512, 515
FLDS	513, 517
float	186
FLOAT	513, 517
float.h	262
float_complex	
- float_complex	400
- float_complex	400
- real	401
- imag	401
- operator=	401
- operator+=	401
- operator-=	401
- operator*=	401
- operator/=	401
- operator=	401
- operator+=	401
- operator-=	401
- operator*=	401
- operator/=	402
float_complex クラス	400
float_complex メンバ外関数	403
floatfield	365
float 型	197
floor	280
floorf	294
FLT_DIG	263
FLT_EXP_DIG	263
FLT_GUARD	262
FLT_MANT_DIG	263
FLT_MAX	262
FLT_MAX_10_EXP	262
FLT_MAX_EXP	262
FLT_MIN	262
FLT_MIN_10_EXP	263
FLT_MIN_EXP	262
FLT_NEG_EPS	263
FLT_NEG_EPS_EXP	264
FLT_NORMALIZE	262
FLT_POS_EPS	263
FLT_POS_EPS_EXP	263
FLT_RADIX	262
FLT_ROUNDS	262
flush	393
FMAC	512, 516

fmod.....	280
fmodf	294
FMOV.....	512, 515
fmtfl	364
fmtflags.....	364
FMUL	512, 516
FNEG.....	512, 516
fopen	305
form	67
fprintf	309
fpcsr	28
FPSCR レジスタ精度モード切り替え.....	28
FPSCR 参照	238
FPSCR 設定	237
fpu.....	31, 733
fputc.....	324
fputs	325
FPU あり指定.....	733
FRCHG.....	517
fread	330
free	342
FreeDir.....	465
FreeFft	448
FreeFir	464
FreeIir	464
FreeLms	465
freopen.....	306
frexp.....	275
frexpf	289
fscanf	314
FSCHG	517
fseek.....	332
fsetpos.....	433
FSQRT.....	516
FSTS	513, 517
FSUB	512, 516
fsy	101
fsymbol	79
ftell.....	333
FTRC	513, 517
ftv	239
FTRV	516
ftvadd.....	240
ftvsub.....	241

FTRV 命令	239
function_call	75
function_forbid	77
fwrite	331
G	
gbr	20
gbr_and_byte	226
gbr_or_byte	226
gbr_read_byte	223
gbr_read_long	224
gbr_read_word	223
gbr_tst_byte	227
gbr_write_byte	224
gbr_write_long	225
gbr_write_word	225
gbr_xor_byte	227
GBR セクション	122
GBR ベースバイト AND	226
GBR ベースバイト OR	226
GBR ベースバイト TEST	227
GBR ベースバイト XOR	227
GBR ベースバイト参照	223
GBR ベースバイト設定	224
GBR ベースロングワード参照	224
GBR ベースロングワード設定	225
GBR パスワード参照	223
GBR パスワード設定	225
GBR ベース変数指定	215
GBR 参照	222
GBR 設定	222
GBR 相対アクセスコード自動生成	20
GE	588, 592
GenBlackman	449
GenGWnoise	475
GenHamming	450
GenHanning	450
GenTriangle	451
get_cr	219
get_fpscr	238
get_gbr	222
get_imask	220
get_vbr	221
getc	326

getchar	326
getenv.....	433
getline	431
gets.....	327
gmtime	433
goodbit.....	365
goptimize	17
GT.....	588, 592
H	
h.....	101
H.....	575
H'.....	488
H'8000 H'8001 置換.....	471
h16	69
h20	69
h32	69
head.....	90
hex	101, 365, 373
hexadecimal	67
HEX ファイル形式.....	739
HIGH 演算.....	498
HLNK_DIR	99
HLNK_LIBRARY1.....	98
HLNK_LIBRARY2.....	98
HLNK_LIBRARY3.....	98
HLNK_TMP.....	99
HUGE_VAL.....	267, 281
HWORD 演算.....	499
I	
IfftComplex.....	441
IfftInComplex	445
IfftInReal.....	446
IfftReal.....	442
ifthen.....	21
ii.....	101
Iir	456
IIR.....	456
Iir1	457
IIR 作業領域解放.....	464
IIR 作業領域割り付け.....	462
imag.....	406, 413
in.....	365

include	6, 15, 40
information	86
init_cnt	363
InitDir	463
InitFft	448
InitFir	462
InitIir	462
InitLms	463
inline	13, 21, 22
in-place FFT	438
in-place 実数 FFT	444
in-place 実数逆 FFT	446
in-place 複素数 FFT	443
in-place 複素数逆 FFT	445
input	63
int	186
INT_MAX	265
INT_MIN	265
int_type	362
internal	365, 372
iomanip	362
ios	90, 362
- ios	368
- ios	368
- init	368
- ~ios	368
- operator void	369
- operator!	369
- rdstate	369
- clear	369
- setstate	369
- good	369
- eof	369
- bad	369
- fail	369
- tie	370
- tie	370
- rdbuf	370
- rdbuf	370
- copyfmt	370
ios_base	
- Init クラス	363
- Init	
Init	363
- Init	

~Init.....	363
- fmtflags.....	365
- iostate.....	365
- openmode	365
- seekdir	366
- _ec2p_init_base	366
- _ec2p_copy_base.....	366
- ios_base	366
- ~ios_base	366
- flags	366
- flags	366
- setf	366
- setf	366
- unsetf	366
- fill	366
- fill	367
- precision	367
- precision	367
- width.....	367
- width.....	367
ios_base クラス.....	364
iostate.....	364
iostream.....	362
ios クラス.....	368
ios クラスマニピュレータ	371
isalnum.....	255
isalpha.....	256
isctrl	256
isdigit	257
isgraph	257
islower	258
isprint.....	258
ispunct.....	259
isspace.....	259
istream.....	362
- sentry クラス	380
- sentry	
sentry	380
- sentry	
~sentry.....	380
- sentry	
operator bool	380
- _ec2p_getistr.....	382
- istream	382
- ~istream	383
- operator>>	383

LDBL_MAX.....	262
LDBL_MAX_10_EXP.....	262
LDBL_MAX_EXP.....	262
LDBL_MIN.....	262
LDBL_MIN_10_EXP.....	263
LDBL_MIN_EXP.....	263
LDBL_NEG_EPS.....	263
LDBL_NEG_EPS_EXP.....	264
LDBL_POS_EPS.....	263
LDBL_POS_EPS_EXP.....	263
LDC.....	510, 514, 517, 518
ldexp.....	276
ldexpf.....	290
ldiv.....	347
ldiv_t.....	336
LDRE.....	518
LDRS.....	518
LDS.....	510, 513, 517, 518
ldtlb.....	232
LDTLB.....	514
LDTLB 命令.....	232
LE.....	588, 592
left.....	365, 372
length.....	15, 422
lib.....	101
library.....	63, 67
Limit.....	471
limits.h.....	265
LIN.....	570
lines.....	59
lis.....	101
list.....	47, 72
LIST.....	566
listfile.....	14
literal.....	44
little.....	30, 54, 734
LITTLE.....	563
Little Endian.....	194
Lms.....	460
Lms1.....	461
locale.h.....	433
localeconv.....	433
localtime.....	433
LOCATE.....	534

log	276, 407, 414
log10	277, 407, 414
log10f	291
logf	290
LogMagnitude	447
logo	36, 60, 86
long	186
long double	186
long double 型	198
LONG_MAX	265
LONG_MIN	265
longjmp	296
loop	26
LOW 演算	499
lpp	101
lseek	152
lst	101
LT	588, 592
LWORD 演算	499

M

MAC	508, 511
MAC.L 命令	237
MAC.W 命令	236
machine.h	218
machinecode	9
macl	237
macll	237
macsave	25
macw	236
macwl	236
MAC レジスタ保証	25
malloc	343
map	18, 71, 101
math	90
math.h	267
mathf	90
mathf.h	281
MatrixMult	476
MaxI	480
mblen	433
mbstowcs	433
mbtowc	433
Mean	478

memchr	354
memcmp	352
memcpy.....	350
memmove	360
memset.....	359
message.....	7, 71
MinI	480
mktime	433
modf.....	277
modff	291
mot	101
MOV	507
MOVA	507
MOVS.L	531
MOVS.W	530, 531
MOVT	507
MOVX.W	530
MOVY.W	530
MsPower	477
mtrx4mul	243
mtrx4muladd.....	244
mtrx4mulsub.....	244
MUL	511
MULS	508
MULU	508
myfptr	396
mystdbuf.....	396
N	
NE.....	588, 592
nearest.....	31, 54
NEG	508
NEGC	508
nest.....	24
new	90, 398
new_handler.....	398
noalign16	25
noboolalpha.....	371
NOCALL	567
NOCODE.....	567
nocompress	83
NOCOND	567
NOCREF.....	566
nocross_reference	50

NODBG	560
nodebug	10, 43, 69
NODEF	567
noexception.....	33
NOEXP	567
noexpansion	15
nofloat	91
noinclude	15
noinline	22
nolist	47
NOLIST	566
nolistfile	14
nologo	36, 60, 86
noloop	26
nomask	13
nomessage.....	7, 71
none	12
nonest.....	24
NOOBJ	560
noobject	15, 45
nooptimize	75
nop	232
NOP	510
noprelink	65
NOPX	530
NOPY	530
NOP 命令	232
norm.....	406, 413
nortnext.....	26
NOSCT	566
nosection	51
noshow	49
noshowbase.....	371
noshowpoint.....	372
noshowpos	372
noskipws	372
nosource.....	15, 48
nospeed	17
NOSRC	566
nostatistics.....	15
NOT	509
not-in-place FFT	438
not-in-place 实数 FFT	440
not-in-place 实数逆 FFT	442

not-in-place 複素数 FFT	439
not-in-place 複素数逆 FFT	441
NOTOPN	266, 686
nouppercase	372
npos	416
NULL	249, 252
O	
obj	101
OBJ	560
object	15, 45, 67
objectfile	11
OCBI	517
OCBP	517
OCBWB	517
oct	365, 373
<u>off</u>	32, 34, 55
OFF	562, 567
off_type	362
offsetof	252
ok_	380, 389
on	32, 34, 55
ON	562, 567
open	150, 396
openmode	364
operator-	405, 412
operator delete	399
operator new	398, 399
operator!=	405, 413, 430
operator*	405, 412
operator/	405, 412
operator+	405, 412, 430
operator<	430
operator<<	391, 394, 406, 413, 431
operator<=	431
operator==	405, 413, 430
operator>	430
operator>=	431
operator>>	388, 406, 413, 431
optimize	16, 75
OR	509
ostream	362
- sentry クラス	389
- sentry	

sentry	389
- sentry	
~sentry	389
- sentry	
operator bool	389
- ostream	391
- ~ostream	391
- operator<<	391
- put	391
- write	391
- flush	391
- tellp	392
- seekp	392
- seekp	392
ostream クラス	390
ostream クラスマニピュレータ	393
ostream メンバ外関数	394
OS システムコール	230
out	365
outcode	37, 58
output	70, 90
overflow	396
P	
p	101
P	121
PABS	528
PADD	528
PADDC	528
PAND	528
patch	27
path	97
pbackfail	396
PCLR	528
PCMP	528
PCOPY	528
PDEC	528
PDMSB	528
PeakI	481
peripheral	13
perror	335
pi	101
pic	32
PINC	528
PLDS	528

PMULS	528
PNEG	528
polar	406, 414
pool	44
POR	528
pos_type	362
pow	278, 407, 414
PowerON_Reset.....	136, 144
powf	292
pp	101
prec	364
PREF.....	513
prefetch	231
PREF 命令.....	231
preinclude	6
preprocessor.....	9
printf	317
PRND.....	528
pro.....	101
profile.....	76
program.....	10
PSHA	528
PSHL	528
PSTS	528
PSUB	528
PSUBC.....	528
ptrdiff_t.....	252
PTRERR	266, 685
putc	327
putchar	328
puts	328
PXOR.....	528
Q	
Q	575
Q'.....	488
qNAN.....	490
qsort	345
R	
raise.....	433
ram.....	80
RAM.....	131
rand	341

RAND_MAX.....	336
read	151
real	406, 413
realloc	343
record	69
reent	91
reference	73
register	75
rel	101
relocate.....	67
remove	433
rename.....	83, 433
REPEAT	617
replace.....	84
resetiosflags	395
return.....	44
rewind	333
right.....	365, 373
rom.....	70, 80
ROM	131
ROM 化支援	70
ROTCL	509
ROTCR.....	509
ROTL.....	509
ROTR.....	509
round.....	31, 54
RTE.....	510
rti.....	101
rtnext.....	26
RTS.....	510
rtti	34
run	12
runtime.....	13, 21, 22, 90
S	
S.....	122, 503, 549, 550, 556
s_len.....	416
s_ptr	416
s_res.....	416
s1	69
s2	69
s3	69
s9.....	82
safe.....	28, 75

same_code.....	75
samecode_forbid.....	77
samesize.....	76
sb.....	368
sbrk.....	153
scanf.....	317
SCHAR_MAX.....	265
SCHAR_MIN.....	265
scientific.....	365, 373
SCT.....	566
sdebug.....	69
section.....	10, 51, 73
SEEK_CUR.....	300, 332
SEEK_END.....	300, 332
SEEK_SET.....	300, 332
seekdir.....	364
seekoff.....	396
seekpos.....	396
set_cr.....	219
set_fpscr.....	237
set_gbr.....	222
set_imask.....	220
set_new_handler.....	399
set_vbr.....	221
setbase.....	395
setbuf.....	307, 396
setfill.....	395
setiosflags.....	395
setjmp.....	296
setjmp.h.....	295
setlocale.....	433
setprecision.....	395
SETRC.....	518
SETS.....	514
SETT.....	510
setvbuf.....	308
setw.....	395
sh1.....	30, 53
SH1.....	97, 533
sh2.....	30, 53
SH2.....	97, 533
sh2e.....	30, 53
SH2E.....	97, 533
sh3.....	30, 53

SH3	97, 533
sh3dsp	53
SH3DSP	97, 533
sh3dspnb.lib	434
sh3dspnl.lib	434
sh3dspnb.lib	434
sh3dsppl.lib	434
sh4	30, 53
SH4	97, 533
SH7055 不具合回避	27
SHAD	513
SHAL	509
SHAR	509
SHC_INC	98
SHC_LIB	97
SHC_TMP	98
SHCPU	97
shdsp	53
SHDSP	97, 533
shdsplib.lib	434
shdspic.lib	434
shift	21
SHLD	513
SHLL	509
SHLL16	509
SHLL2	509
SHLL8	509
SHLR	509
SHLR16	509
SHLR2	509
SHLR8	509
short	186
show	15, 49, 73
showbase	365, 371
showmanyc	396
showpoint	365, 372
showpos	365, 372
SHRT_MAX	265
SHRT_MIN	265
signal	433
signal.h	433
signal_sem	154
signed char	186
sin	272, 407, 414

sinf	286
single	31
sinh	274, 407, 414
sinhf	288
size	17
size_t	252
SIZEOF 演算	497
sjis	36, 37, 57, 58
skipws	365, 372
sleep	229
SLEEP	510
SLEEP 命令	229
smachine.h	218
smanip クラス マニピュレータ	395
sNAN	490
sni	101
source	15, 48
sp	206, 213
speed	17, 75
sprintf	318
sputc	377
sqrt	278, 407, 414
sqrtf	292
srnd	341
src	101
SRC	566
SR 参照	219
SR 設定	219
sscanf	318
st_ext	245
stack	82, 121
STACK	534
start	78
STARTOF 演算	497
state	368
static	12
static ラベル名	728
statistics	15
STC	510, 514, 517, 518
stdarg	90
stdarg.h	297
stddef.h	252
stderr	249, 300
stdin	249, 300

stdio	90
stdio.h	300, 433
stdlib	90
stdlib.h	336, 433
stdout	249, 300
strcat.....	351
strchr	354
strcmp	353
strcoll	433
strcpy	350
strcspn.....	355
streambuf	362
- streambuf	376
- ~streambuf	376
- pubsetbuf	376
- pubseekoff	376
- pubseekpos	376
- pubsync.....	376
- in_avail	376
- snextc.....	376
- sbumpc.....	377
- sgetc.....	377
- sgetn	377
- sputbackc.....	377
- sungetc.....	377
- sputn	377
- eback.....	377
- gptr	378
- egptr.....	378
- gbump.....	378
- setg	378
- pbase.....	378
- pptr	378
- epptr.....	378
- pbump.....	378
- setp	378
- setbuf	378
- seekoff	379
- seekpos	379
- sync.....	379
- showmanyc.....	379
- xsgetn	379
- underflow.....	379
- uflow.....	379
- pbackfail	379
- xsputn	379

- overflow.....	379
streambuf クラス	374
streamoff	362
streamsize	362
strerror	359
strftime.....	433
string	11, 90, 416
- string.....	421
- string.....	421
- string.....	421
- string.....	421
- string.....	421
- ~string.....	421
- operator=	421
- operator=	421
- operator=	421
- begin	421
- end	422
- size.....	422
- max_size.....	422
- resize.....	422
- resize.....	422
- capacity.....	422
- reserve	422
- clear	422
- empty	422
- operator.....	423
- at	423
- operator+=	423
- operator+=	423
- operator+=	423
- append	423
- append	423
- append	423
- append	423
- assign.....	423
- assign.....	423
- assign.....	424
- assign.....	424
- insert	424
- insert	424
- insert	424
- insert	424
- insert	424
- insert	424
- insert	424
- insert	424
- erase.....	424

- erase.....	424
- erase.....	425
- replace.....	425
- replace.....	425
- replace.....	425
- replace.....	425
- replace.....	425
- replace.....	425
- replace.....	425
- copy.....	425
- swap.....	425
- c_str.....	425
- data.....	425
- find.....	426
- find.....	426
- find.....	426
- rfind.....	426
- rfind.....	426
- rfind.....	426
- find_first_of.....	426
- find_first_of.....	426
- find_first_of.....	426
- find_last_of.....	426
- find_last_of.....	427
- find_last_of.....	427
- find_first_not_of.....	427
- find_first_not_of.....	427
- find_first_not_of.....	427
- find_last_not_of.....	427
- find_last_not_of.....	427
- find_last_not_of.....	427
- substr.....	427
- compare.....	427
- compare.....	428
- compare.....	428
- compare.....	428
- compare.....	428
string.h.....	348, 433
string_unify.....	75
string クラス.....	416
string クラス マニピュレータ.....	429
strip.....	85
strlen.....	360
strncat.....	352
strncmp.....	353
strncpy.....	351

strpbrk.....	355
strchr.....	356
strspn.....	356
strstr.....	357
strtod.....	339
strtok.....	358
strtol.....	340
strtoul.....	433
strxfrm.....	433
STS.....	510, 513, 517, 518
stype.....	67
SUB.....	508
sub4.....	242
SUBC.....	508
subcommand.....	37, 60, 88
SUBV.....	508
swap.....	431
SWAP.....	507
SWAP.B 命令.....	235
SWAP.W 命令.....	235
swapb.....	235
swapw.....	235
switch 文展開方式.....	21
symbol.....	73
symbol_delete.....	75
symbol_forbid.....	77
sync.....	396
SYS_OPEN.....	300
sysrof.....	735
system.....	433
S タイプファイル形式.....	737

T

tab.....	15
table.....	21
tan.....	273, 407, 415
tanf.....	287
tanh.....	274, 407, 415
tanhf.....	288
tas.....	229
TAS.....	509
TAS 命令.....	229
td.....	101
template.....	12

ti	101
tiestr	368
time	433
time.h	433
TMP_MAX	300
tmpfile	433
tmpnam	433
tn	206
tolower	261
toupper	261
tpldir	101
trace	231
TRACE 命令	231
trapa	230
TRAPA	510
trapa_svc	230
TRAPA 命令	230
trunc	365
TST	509
U	
UCHAR_MAX	265
UINT_MAX	265
ULONG_MAX	265
umachine.h	218
underflow	396
ungetc	329
unitbuf	365
unsigned char	186
unsigned int	186
unsigned long	186
unsigned short	186
uppercase	365, 372
used	12
user	20
USHRT_MAX	265
V	
va_arg	298
va_end	299
va_list	297
va_start	298
value_type	400, 408
variable_access	75

variable_forbid.....	77
Variance.....	479
VBR 参照.....	221
VBR 設定.....	221
VEC_TBL.....	136, 137
VectorMult.....	477
vfprintf.....	319
volatile_loop.....	28
vprintf.....	320
vsprintf.....	321
W	
W.....	503, 543, 544, 551, 552
wait_sem.....	154
warning.....	56, 86
wcstombs.....	433
wctomb.....	433
wide.....	364
width.....	15
write.....	152
ws.....	387
X	
XOR.....	509
XTRCT.....	507
X データ転送オペレーション.....	523
X データ転送命令.....	530
X メモリ Y メモリコピー.....	472
X メモリからコピー.....	474
X メモリへコピー.....	473
Y	
Y データ転送オペレーション.....	523
Y データ転送命令.....	530
Y メモリ X メモリコピー.....	472
Y メモリからコピー.....	474
Y メモリへコピー.....	473
Z	
zero.....	31, 54

日本語

ア行	
アセンブラ	2
アセンブラ記述関数インライン展開	210
アセンブラ言語仕様	483
アセンブラ制御命令	532
アSEMBルリスト	109
アSEMBルリスト桁数設定	59
アSEMBルリスト行数/桁数設定	570
アSEMBルリスト行数設定	59
アSEMBルリスト出力制御	47, 566
アドレスシンボル	486
アドレス形式	503, 526, 529
アドレス整合性チェック	80
余り	280, 294
アングフロー	200
イミディエイト	504
インクリメント	504
インクルードファイルの基点	724
インクルードファイルディレクトリ	6, 40
インターナル	621, 707
インデックス付き GBR 間接	504
インデックス付きレジスタ間接	504
インデックス付きレジスタ間接/ポストインクリメント	529
インフォメーション	621, 707
インフォメーションメッセージ	7, 71
ウォーニング	621, 687, 707, 719
エラー	621, 687, 707, 719
エラーメッセージ	621, 687, 707, 719
エラーメッセージ出力	335
エラーメッセージ文字列	359
エラーレベル	621, 687, 707, 719
エラーレベル変更	56
エラー形式	621, 687, 707, 719
エラー指示子	251
エラー情報	114, 118
エラー状態クリア	334
エンディアン指定	734
エントリ関数作成	213
オーバフロー	200
オフセットの種類	332
オブジェクトファイル	731
オブジェクトファイル形式	731

オブジェクトファイル名指定.....	11
オブジェクトモジュール、デバッグ情報出力制御.....	560
オブジェクトモジュール出力制御.....	45
オブジェクトモジュール名設定.....	574
オブジェクト形式.....	9, 724
オブジェクト情報.....	105
オブジェクト内漢字コード指定.....	37
オプション情報.....	113, 117
オペランド.....	483
オペレーション.....	483
オペレーションサイズ.....	503, 526, 529
位置指示子.....	251
位置指定.....	603
印字文字.....	254
印字文字判定.....	258
英字.....	254
英字、10進数字判定.....	255
英字判定.....	256
英小文字.....	254
英小文字判定.....	258
英小文字変換.....	261
英大文字.....	254
英大文字判定.....	260
英大文字変換.....	261
演算の順序.....	496
演算子.....	495
演算子の評価順序.....	202
欧州コード文字.....	57
カ行	
カッコ.....	496
ガードビット.....	250
キーワード指定.....	603
キャッシュサイズ.....	77
クラス.....	184
クラス型.....	187
クロスリファレンスリスト.....	111
クロスリファレンスリスト出力制御.....	50
グローバルクラスオブジェクト後処理.....	137
グローバルクラスオブジェクト初期処理.....	137
グローバル変数レジスタ割り付け.....	214
コピーライト出力制御.....	36, 60, 86
コマンドラインインタフェース.....	725
コマンド指定情報.....	108

コメント	483
コメントのネスト	24
コンディション	523
コンディション付き DSP 演算命令	527
コンパイラ	2
コンパイラの暗黙の宣言	100
コンパイルリスト	102
仮引数	598
仮引数のデフォルト	598
仮数、指数を浮動小数点数に変換	276, 290
仮数部	196
可変個パラメタファイル出力	319
可変個パラメタ出力	320
可変個パラメタ文字列出力	321
可変個引数取り出し	298
可変個引数取り出し開始	298
可変個引数取り出し終了	299
回転係数解放	448
回転係数生成	448
開始アドレス	617
外部シンボル割り付け情報ファイル出力	71
外部参照シンボル宣言	558
外部定義シンボル、外部参照シンボル宣言	559
外部定義シンボル宣言	557
外部変数アクセス最適化	18
拡張レジスタからのストア	245
拡張レジスタへのロード	245
拡張機能	203
完全畳み込み	466
環境	181
環境変数	97
簡易入出力関数	91
関数	248
関数アクセス最適化対象シンボル情報	116
関数インライン展開	209
関数メンバへのポインタ	186
基数	250
基数指定	575
基本型	186
記憶域移動	360
記憶域解放	342
記憶域割り当て	343
記憶域割り当てサイズ変更	343
記憶域内文字検索	354

記憶域比較	352
記憶域複写	350
擬似乱数生成	341
擬似乱数列初期設定	341
逆正弦	269, 283
逆正接	270, 284
逆余弦	269, 283
旧バージョンとの互換性	724
共通コードサイズ	76
共用体	184
共用体型	187
境界調整数	121
空白を除く印字文字判定	257
空白類文字	254
空白類文字判定	259
偶奇演算	499
繰り返し展開	581, 591
係数のスケーリング	452
型変換の規則	170
形式種別	121
継続処理	87
計数付き文字列データ確保	547
計数付き文字列データ領域確保	554
結合規則	496
結合性	202
言語仕様	181
限界値	721
固定小数点データ確保	551
固定小数点データ形式	493
固定小数点定数	492
効率	436
構造体	184
構造体型	187
行番号変更	565
項	495
高速フーリエ変換	437
書き込み操作	121
切り捨て	280, 294
切り上げ	279, 293
組み込み関数	217
組み込み向け C++ 言語	24
サ行	
サイズモード	609

サイズ指定モード.....	609
サイズ選択モード.....	609
サブコマンドファイル指定.....	37, 60, 88
システム制御命令.....	510, 513, 514, 517, 518
シフト JIS.....	57
シフト演算展開.....	21
シフト演算命令.....	513
シフト命令.....	509
シングルデータ転送オペレーション.....	523
シングルデータ転送命令.....	530, 531
シングルメモリ転送.....	529
シンボル.....	486
シンボルアドレスファイル.....	79
シンボルデバッグ情報部分出力制御.....	562
シンボル削除最適化情報.....	116
シンボル指定による PC 相対.....	504
シンボル情報.....	115
シンボル値設定.....	539, 540
シンボル定義.....	64
シンボル名削除.....	84
シンボル名変更.....	83
スカラー型.....	186
スケーリング.....	438
スタックセクション作成.....	205
スタック解析ツール.....	3, 93
スタック情報ファイル.....	82
スタック切り替え指定.....	206
スタック領域.....	122, 132
ストリーム入出力.....	248
ストリーム入出力用クラスライブラリ.....	362
スピード・サイズ最適化.....	17
セクション.....	121
セクションの結合.....	125
セクションの初期化.....	137
セクションアドレス演算子.....	216
セクション開始アドレス.....	78
セクション初期化用テーブル.....	136, 145
セクション情報リスト.....	112
セクション情報リスト出力制御.....	51
セクション切り替え指定.....	204
セクション宣言.....	534
セクション属性.....	121
セクション名.....	10, 486
セマフォ解放.....	154

セマフォ確保	154
ゼロ	196, 198, 199, 490
ゼロ終端文字列データ確保	548
ゼロ終端文字列データ領域確保	555
ソースステートメント	483, 523
ソースプログラムの終わりとエントリポイント指定	576
ソースプログラムリストヘッダ設定	571
ソースプログラムリスト改ページ	572
ソースプログラムリスト空行出力	573
ソースプログラムリスト出力制御	48
ソースプログラムリスト部分出力制御	49, 567
ソースリスト情報	102, 110
ソート	345
最後の文字位置	356
最初の文字位置	354
最初の文字列位置	357
最小値	480
最大絶対値	481
最大値	480
最適化	75
最適化リンケージエディタ	2
最適化レベル	16
最適化部分抑止	77
作業領域	452
三角窓	451
算術演算命令	508, 511, 512, 516
指数関数	275, 289
指数部	196, 491
指定インクルードファイル取り込み	578
指定文字群が最初に現れるまでの文字数	355
指定文字群が最初に現れる位置	355
指定文字群が連続する部分の長さ	356
字句切り分け	358
自然対数	276, 290
自動インライン展開	22
式	495
式の要素	495
識別子	181
実行開始アドレス	65
実行環境の設定	136
実行時型情報	34
実行命令	503
実数データ配列フォーマット	437
周期的畳み込み	467

周期的相関	470
修飾子	184
終端コード	82
終了アドレス	617
終了指定	87
終了処理ルーチン	137, 162
出力ファイル	70, 90
出力漢字コード	58
出力形式	67
処理系定義	251
初期化データ領域	121
初期化付き記憶域割り当て	342
初期設定	136, 144
初期設定プログラムの作成	128
初期値	121
書式付きファイル出力	309
書式付きファイル入力	314
書式付き出力	317
書式付き入力	317
書式付き文字列出力	318
書式付き文字列入力	318
除算後の逆正接	271, 285
除算方式選択	13
商と余り	346, 347
小数部	491, 493
乗算	477
常用対数	277, 291
条件つきアセンブリ機能	579
条件つきアセンブル	580
条件つき繰り返し展開	581, 592
畳み込み	466
診断	253
制御文字	254
制御文字判定	256
整数	182
整数データブロック確保	544
整数データ確保	543
整数型プリプロセッサ変数	579
整数型プリプロセッサ変数定義	41, 584
整数定数	488
正規化	250
正規化数	196, 197, 198, 490
正弦	272, 286
正接	273, 287

精度	310
静的領域	129
絶対アドレス形式	121
絶対値	279, 293, 345, 346
宣言	184
双曲線正弦	274, 288
双曲線正接	274, 288
双曲線余弦	273, 287
相関	466, 469
相対アドレス形式	121
タ行	
テキストファイル	249
テンプレートインスタンス生成機能	12
データの内部表現	186
データフォーマット	434
データメンバへのポインタ	186
データ書き出し	152
データ転送命令	507, 512, 513, 515, 524, 529
データ転送命令一覧	530
データ読み込み	151
データ領域確保	552
ディスプレースメント	504
ディスプレースメント付き GBR 間接	504
ディスプレースメント付き PC 相対	504
ディスプレースメント付きレジスタ間接	504
デクリメント	504
デバッグ情報	10, 43, 69
デバッグ情報圧縮	83
デバッグ情報削除	85
デフォルトインクルードファイル	6
デュアルメモリ転送	529
トラップ命令リターン指定	206
対象ヘッダファイル	90
対数絶対値	447
大域 goto	296
大域 goto 飛び先設定	296
単一データ用 FIR	455
単一データ用 IIR	457
単一データ用適応 FIR	461
単一データ用倍精度 IIR	459
置換シンボル	579
置換シンボル定義	40
低水準インタフェースルーチン	137, 148

定義域エラー	268, 282
定義型条件つきアセンブル.....	580, 590
定数	488
定数シンボル	486
定数領域	121
適応 FIR	460
適応 FIR 作業領域解放	465
適応 FIR 作業領域割り付け	463
展開の中断終了	593
展開上限値設定	594
転送コード展開	22
統計情報	107
動的領域	132
特殊文字	254
特殊文字判定	259
ナ行	
二分割検索	344
入力ファイル	63
ニーモニック	503, 529
ヌル文字	249
ハ行	
べき乗	278, 292
ハニング窓	450
ハミング窓	450
バイナリファイル.....	64, 249
バッファフラッシュ.....	304
バッファリング制御.....	308
バッファ領域設定.....	307
パラメタのサイズ指定.....	310
ヒープ領域	122, 134
ビットフィールド.....	184, 192
ファイルから 1 文字入力.....	322, 326
ファイルから文字列入力.....	323
ファイルに 1 文字出力.....	324, 327
ファイルに 1 文字返却.....	329
ファイルに文字列入力.....	325
ファイルのクローズ.....	137, 147
ファイルアクセスモード.....	250
ファイルインクルード機能.....	577
ファイルエラー状態判定.....	335
ファイルオープン	150, 305
ファイルクローズ.....	151, 304

ファイルポインタ	248
ファイル拡張子	101
ファイル再オープン	306
ファイル仕様	101
ファイル終了指示子	251
ファイル終了判定	334
ファイル書き込み	331
ファイル先頭に移動	333
ファイル読み込み	330
ファイル読み書き位置移動	332
ファイル読み書き位置取得	333
ファイル内位置設定	152
フィールド幅	310, 314
フィルタ	452
フェータル	621, 687, 707, 719
フォーマットコンバータ	3, 731
フラグ	309
ブラックマン窓	449
プリデクリメントレジスタ間接	504, 529
プリプロセッサ	185
プリプロセッサ置換文字列定義	587
プリプロセッサ展開	9
プリプロセッサ展開結果出力	44
プリプロセッサ展開時エラー処理	594
プリプロセッサ変数	579
プレリンカ	3
プレリンカ起動抑止	65
プログラミング	121
プログラムの開発手順	1
プログラムの構造	121
プログラムの動作保証	723
プログラムの要素	523
プログラム領域	121
プロファイル情報	76
ベクタテーブルの設定	136, 137
ポインタ	183, 186
ポジションインディペンデントコード	32
ポストインクリメントレジスタ間接	504, 529
引数の割り付け	172
引数の割り付け領域	171
引数の渡し方	169
配列	183
配列型	187
倍精度 IIR	458

倍精度 IIR 作業領域解放	465
倍精度 IIR 作業領域割り付け	463
白色ガウス雑音	475
範囲エラー	268, 282
比較型条件つきアセンブル	580, 588
非数	197, 198, 199, 490
非正規化数	196, 197, 199, 490
非正規化数の扱い	32, 55
標準 C ライブラリ	246
標準エラー出力ファイル	249
標準ライブラリ構築ツール	3
標準出力ファイル	249
標準入出力ファイル	249
標準入力ファイル	249
浮動小数点	182
浮動小数点データブロック確保	550
浮動小数点データ確保	549
浮動小数点データ形式	490
浮動小数点データ領域確保	556
浮動小数点レジスタ別名定義	542
浮動小数点演算モード	31
浮動小数点数	250
浮動小数点数の仕様	196
浮動小数点数の内部表現	196
浮動小数点数を仮数、指数に分解	275, 289
浮動小数点数を整数部、小数部に分解	277, 291
浮動小数点定数	489
浮動小数点定数除算の乗算化	27
符号部	196, 491, 493
部分的畳み込み	468
複合型	187
複素数データ配列フォーマット	437
複素数計算用クラスライブラリ	400
分岐命令	510, 511
文	184
文字	181
文字の種類	254
文字繰り返し	359
文字型プリプロセッサ変数	579
文字型プリプロセッサ変数定義	42, 586
文字定数	488
文字列	501
文字列の検索	606
文字列の切り出し	607

文字列の長さ	360, 605
文字列を double 型に変換	337, 339
文字列を int 型に変換	337
文字列を long 型に変換	338, 340
文字列データブロック確保	546
文字列データ確保	545
文字列データ領域確保	553
文字列出力	328
文字列出力領域	11
文字列操作関数	602, 605
文字列操作用クラスライブラリ	416
文字列内文字コード	36
文字列入力	327
文字列比較	353
文字列複写	350, 351
文字列連結	351, 352
平均	479
平均 2 乗値	477
平均値	478
平方根	278, 292
並列演算命令	523
偏差	479
変換形式指定	735
変換仕様	309, 314
変換文字	311, 315
変数アクセス最適化対象シンボル情報	116
マ行	
マクロ	248
マクロコール	596, 603
マクロ機能	595
マクロ処理除外	601
マクロ生成番号	601
マクロ定義	596
マクロ内コメント	602
マクロ本体	600
マクロ名	598
マクロ名定義	7
マクロ命令定義	598
マトリックス乗算	476
メッセージレベル	86
メモリのバイト並び順	30, 54, 563
メモリ管理用ライブラリ	398
メモリ領域の割り付け	129

メモリ領域割り付け.....	153
モジュール間最適化.....	17
モジュール置換.....	84
モジュール抽出.....	85
丸め.....	250
丸め方式.....	31, 54
窓関数.....	449
未サポートライブラリ.....	433
未初期化データ領域.....	121
無限大.....	197, 198, 199, 490
無効演算.....	200
ヤ行	
優先順位.....	202, 496
予約語.....	485
余弦.....	272, 286
ラ行	
ライブラリファイル.....	63, 731
ライブラリリスト.....	117
ライブラリ情報.....	118
ライブラリ内モジュール、セクション、シンボル情報.....	119
ラベル.....	483
ラベルの 16 バイト整合.....	25
リエントラントライブラリ.....	91, 432
リストファイル.....	14, 72
リスト内容.....	73
リスト内容と形式.....	15
リターンコード.....	249
リターン値の設定場所.....	172
リターン値拡張.....	26
リテラル.....	608
リテラルプール.....	608
リテラルプール自動生成機能.....	608
リテラルプール自動生成機能サイズモード設定.....	52
リテラルプール出力ポイント指定.....	44
リピートループ命令自動生成機能.....	616
リピート回数.....	617
リピート制御命令.....	518
リファレンス.....	186
リンケージマップ情報.....	114
リンケージリスト.....	113
ループ展開最適化.....	26
ループ判定式最適化抑止.....	28

レコードサイズ統一.....	69
レジスタ.....	183
レジスタ間接.....	504, 529
レジスタ退避 / 回復コード制御機能.....	212
レジスタ直接.....	504
レジスタ別名.....	486
レジスタ別名定義.....	541
ローカルラベル.....	501
ロードモジュールファイル.....	731
ロケーションカウンタ.....	494
ロケーションカウンタ値設定.....	537
ロケーションカウンタ値補正.....	538
ロングワードサイズ固定小数点定数.....	493
例外処理機能.....	33
列挙型.....	184
論理演算命令.....	509
ワ行	
ワードサイズ固定小数点定数.....	492
割り込みマスク参照.....	220
割り込みマスク設定.....	220
割り込み関数作成.....	206

SuperH RISC engine C/C++ コンパイラ、アセンブラ、最適化リンケージエディタ
ユーザーズマニュアル



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

ADJ-702-444A