

お知らせ:
175ページの表C.3 仕様 中に誤記の訂正がございます。

R32C/100 シリーズ用
C コンパイラパッケージ V.1.02
C コンパイラユーザーズマニュアル

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用途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ものです。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、海底中継機器、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ン지니어リング処理等、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て、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。

はじめに

NC100 は、ルネサス 32 ビットマイクロコンピュータ R32C/100 シリーズ用の C コンパイラです。NC100 は、C 言語で記述したプログラムを R32C/100 用のアセンブリ言語ソースファイルに変換します。また、コンパイルオプションを指定することによって、アセンブル / リンクを実行してマイクロコンピュータに書き込み可能な 16 進数形式ファイルまでを生成することができます。なお、記載されている注意事項をよくお読みの上、ご使用ください。

- Microsoft、MS-DOS、Windows および Windows NT は、米国 Microsoft Corporation の米国およびその他の国における商標または登録商標です
- IBM および AT は、米国 International Business Machines Corporation の登録商標です。
- Intel、Pentium は、米国 Intel Corporation の登録商標です。
- Adobe および Acrobat は、Adobe Systems Incorporated (アドビシステムズ社) の登録商標です。
- Netscape および Netscape Navigator は、米国およびその他の諸国の Netscape Communications Corporation 社の登録商標です。
- その他すべてのブランド名および製品名は個々の所有者の登録商標もしくは商標です。

1. 用語の使い分けの説明

本ユーザーズマニュアルでは表現上、以下に示す用語を使い分けています。

用語	意味
NC100	R32C/100シリーズ用コンパイラ・パッケージを意味します。
nc100	コンパイルドライバ、又はその実行ファイルを意味します。
AS100	R32C/100シリーズ用コンパイラパッケージに含まれるアセンブラ・パッケージを意味します。
as100	リロケータブルアセンブラ、又はその実行ファイルを意味します。
High-performance Embedded Workshop	付属の統合環境を意味します

2. 使用する記号の説明

NC100 のマニュアルでは、以下に示す記号を使用します。

記号	表示内容
A>	MS-Windows (TM) のプロンプトを示します。
<RET>	リターンキーの入力を示します。
<>	< >の中の部分は必須項目を示します。
[]	[]の中の部分は省略可能であることを示します。
△	スペース又はタブコードを示します (必須)。
▲	スペース又はタブコードを示します (省略可能)。
: (省略) :	ファイルの表示中の省略を示します。

なお、その他の記号を使用するときは、適宜説明します。

第 1 章	NC100 の処理概要	1
1.1	NC100の構成	1
1.2	NC100の処理フロー	2
1.2.1	nc100	3
1.2.2	igen100	3
1.2.3	cpp100	3
1.2.4	ccom100	3
1.2.5	aopt100	3
1.2.6	Call Walker & gensni	3
1.2.7	MapView	3
1.3	注意事項	4
1.3.1	コンパイラのバージョンアップ等についての注意事項	4
1.3.2	マイコンの機種依存部に関する注意事項	4
1.4	プログラム開発例	5
1.5	NC100の出力ファイル	7
1.5.1	出力ファイルの概要	7
1.5.2	プリプロセス結果 C 言語ソースファイル	8
1.5.3	アセンブリ言語ソースファイル	10
第 2 章	コンパイラの基本的な使い方	13
2.1	コンパイラの起動	13
2.1.1	コンパイルドライバのコマンドの入力書式	13
2.1.2	コマンドファイル	14
2.1.3	起動オプションに関する注意事項	15
2.1.4	nc100 の起動オプション	16
2.2	スタートアッププログラムの準備	21
2.2.1	スタートアッププログラムのサンプル	21
2.2.2	スタートアッププログラムのカスタマイズ	33
2.2.3	メモリ配置のカスタマイズ	38
第 3 章	プログラミング	50
3.1	注意事項	50
3.1.1	コンパイラのバージョンアップ等についての注意事項	50
3.1.2	マイコンの機種依存部に関する注意事項	50
3.1.3	最適化について	51
3.1.4	register 変数の使用に関する注意事項	54
3.1.5	スタートアップの扱いについて	54

3.2	生成コードの向上のために.....	55
3.2.1	コード効率の良いプログラミング方法.....	55
3.2.2	スタートアップ処理を高速化する方法.....	56
3.3	アセンブリ言語プログラムとの結合方法.....	57
3.3.1	C 言語プログラムからアセンブラ関数の呼び出し方法.....	57
3.3.2	アセンブラ関数の記述方法.....	60
3.3.3	アセンブラ関数の記述に関する注意事項.....	63
3.4	その他.....	64
3.4.1	NC シリーズコンパイラ間の移植に関する注意事項.....	64
付録 A	コマンドオプションリファレンス.....	65
A.1	コンパイルドライバの入力書式.....	65
A.2	起動オプション.....	66
A.2.1	コンパイルドライバの制御に関するオプション.....	66
A.2.2	出力ファイル指定オプション.....	70
A.2.3	バージョン情報及びコマンドライン表示オプション.....	71
A.2.4	デバッグ用オプション.....	72
A.2.5	最適化オプション.....	73
A.2.6	生成コード変更オプション.....	87
A.2.7	ライブラリ指定オプション.....	95
A.2.8	警告オプション.....	96
A.2.9	アセンブル / リンクオプション.....	105
A.3	起動オプションに関する注意事項.....	106
A.3.1	起動オプションの記述に関する注意事項.....	106
A.3.2	オプションの優先順位.....	106
付録 B	拡張機能リファレンス.....	107
B.1	near/far修飾子.....	109
B.1.1	near/far 修飾子の概要.....	109
B.1.2	変数の宣言書式.....	109
B.1.3	ポインタ型変数の宣言書式.....	110
B.1.4	関数の宣言.....	112
B.1.5	nc100 の起動オプションによる near/far の制御.....	112
B.1.6	near から far への型変換機能.....	113
B.1.7	関数の宣言.....	113
B.1.8	複数の宣言で near/far の確定を行う機能.....	114
B.1.9	near/far 属性に関する注意事項.....	115
B.2	asm関数.....	116

B.2.1	asm 関数の概要	116
B.2.2	auto 変数の FB オフセット値の指定	117
B.2.3	レジスタ変数のレジスタ名の指定	120
B.2.4	extern 変数及び static 変数のシンボル名の指定	121
B.2.5	記憶クラスに依存しない指定	124
B.2.6	最適化の部分的な抑止方法	125
B.2.7	asm 関数に関する注意事項	125
B.3	日本語文字サポート	128
B.3.1	日本語文字の概要	128
B.3.2	日本語文字を記述するための設定	128
B.3.3	文字列中の日本語文字	129
B.3.4	文字定数としての日本語文字	130
B.4	関数のデフォルト引数宣言	131
B.4.1	関数のデフォルト引数宣言の概要	131
B.4.2	関数のデフォルト引数宣言の書式	131
B.4.3	関数のデフォルト引数宣言の規定事項	133
B.5	inline関数宣言	134
B.5.1	inline 記憶クラスの概要	134
B.5.2	inline 記憶クラスの宣言書式	134
B.5.3	inline 記憶クラスの規定事項	135
B.6	コメント "//"の概要	138
B.6.1	コメント "//"の概要	138
B.6.2	コメント "//"の書式	138
B.6.3	"//" と "/*"の優先順序	138
B.7	#pragma 拡張機能	139
B.7.1	#pragma 拡張機能の一覧	139
B.7.2	メモリ配置に関する拡張機能	143
B.7.3	組み込み機器に関する拡張機能の使用法	153
B.7.4	その他の拡張機能の使用法	162
B.8	アセンブラマクロ関数	167
B.8.1	アセンブラマクロ関数の概要	167
B.8.2	アセンブラマクロ関数の記述例	167
B.8.3	アセンブラマクロ関数で記述可能な命令	168
付録 C	C 言語仕様概要	174
C.1	性能仕様	174
C.1.1	標準仕様概要	174
C.1.2	性能概要	174

C.2	基本言語仕様.....	176
C.2.1	文法.....	176
C.2.2	型.....	179
C.2.3	式.....	181
C.2.4	宣言.....	181
C.2.5	文.....	184
C.3	プリプロセスコマンド.....	187
C.3.1	プリプロセスコマンドの機能別一覧.....	187
C.3.2	プリプロセスコマンドリファレンス.....	187
C.3.3	プリデファインドマクロ.....	195
C.3.4	プリデファインドマクロの使用法.....	195
付録 D	C 言語実装仕様.....	196
D.1	データの内部表現.....	196
D.1.1	整数型.....	196
D.1.2	浮動小数点型.....	197
D.1.3	列挙型.....	198
D.1.4	ポインタ型.....	198
D.1.5	配列型.....	198
D.1.6	構造体型.....	199
D.1.7	共用体型.....	199
D.1.8	ビットフィールド型.....	200
D.2	符号拡張規則.....	201
D.3	関数呼び出し規則.....	202
D.3.1	戻り値に関する規則.....	202
D.3.2	引き数渡しに関する規則.....	202
D.3.3	関数のアセンブリ言語シンボルへの変換規則.....	204
D.3.4	関数間のインターフェース.....	208
D.4	auto変数の領域確保.....	213
D.5	レジスタの退避.....	214
付録 E	標準ライブラリ.....	215
E.1	標準ヘッダファイル.....	215
E.1.1	標準ヘッダファイルの概要.....	215
E.1.2	標準ヘッダファイルリファレンス.....	216
E.2	標準関数リファレンス.....	226
E.2.1	標準関数ライブラリの概要.....	226
E.2.2	標準関数ライブラリ機能別一覧.....	227

E.2.3	標準関数リファレンス.....	232
E.2.4	標準関数ライブラリの使用に関する注意事項.....	315
E.3	標準入出力関数ライブラリのカスタマイズ方法.....	316
E.3.1	入出力関数の構成.....	316
E.3.2	入出力関数の変更手順.....	317
付録 F	エラーメッセージ一覧表.....	324
F.1	メッセージの出力形式.....	324
F.2	nc100エラーメッセージ.....	325
F.3	cpp100エラーメッセージ.....	327
F.4	cpp100ワーニングメッセージ.....	330
F.5	ccom100エラーメッセージ.....	331
F.6	ccom100ワーニングメッセージ.....	343
付録 G	Call Walker 用スタック情報ファイル作成ツール gensni 操作方法.....	352
G.1	Call Walkerの起動.....	352
G.2	gensniの概要.....	352
G.2.1	gensni の処理概要.....	352
G.3	gensniの起動方法.....	353
G.3.1	入力書式.....	353
G.3.2	オプションリファレンス.....	355
G.4	gensniのエラーメッセージ.....	356
G.4.1	エラーメッセージ.....	356

第1章 NC100 の処理概要

この章では、NC100 が行うコンパイル処理の概要と、NC100 を使用したプログラム開発の事例を説明します。

1.1 NC100 の構成

NC100 は、以下に示す7つの実行ファイルで構成されています。

- | | | |
|-----|----------------------|---------------------------|
| (1) | nc100 | コンパイルドライバ |
| (2) | igen100 | インラインジェネレータ |
| (3) | cpp100 | プリプロセッサ |
| (4) | ccom100 | コンパイラ |
| (5) | aopt100 | アセンブラオブティマイザ |
| (6) | Call Walker & gensni | スタック解析ツール&スタック情報解析ユーティリティ |
| (7) | MapView | マップビューワ |

1.2 NC100 の処理フロー

NC100 の処理フローを【図 1.1】に示します。

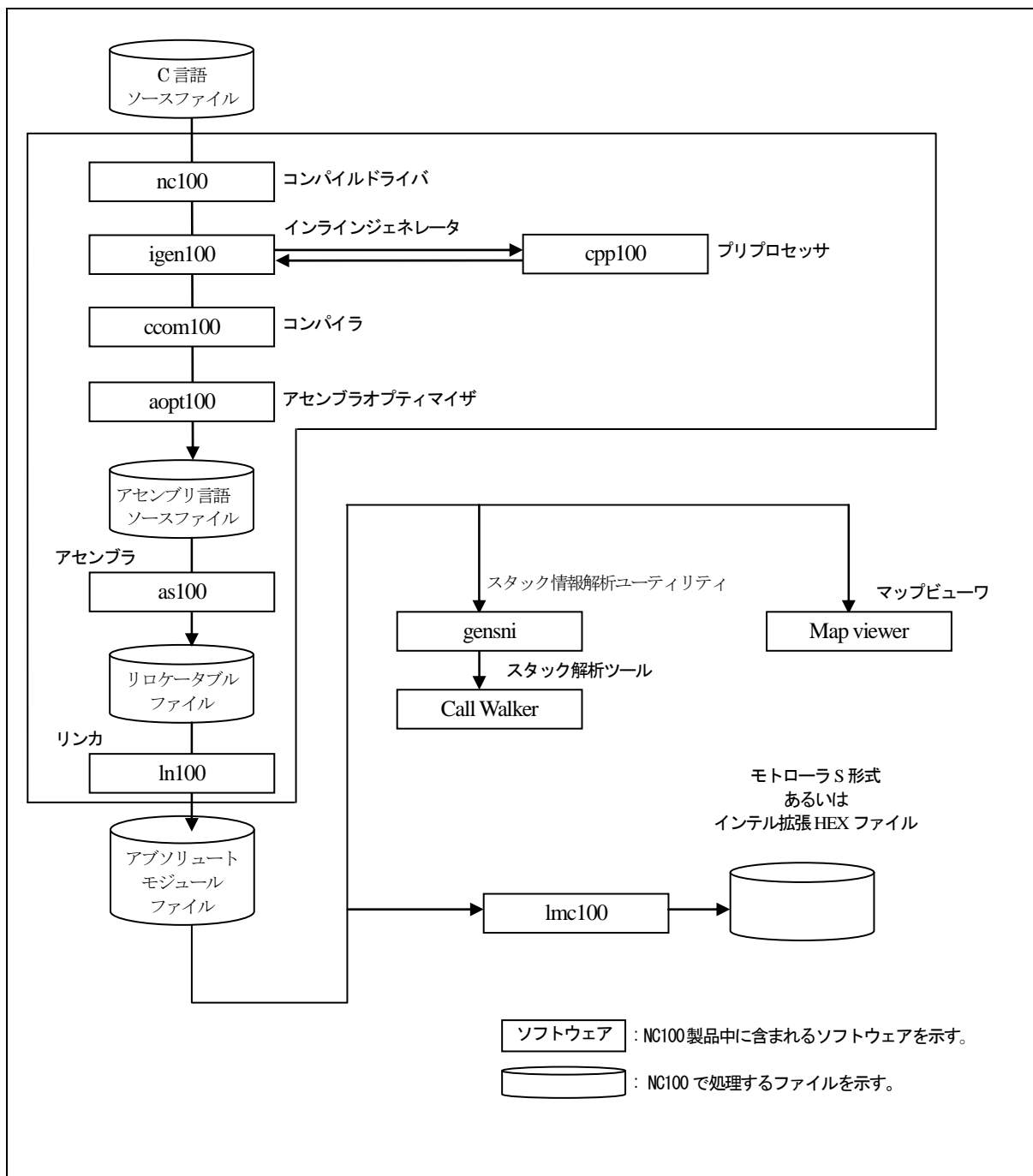


図1.1 NC100の処理フロー

1.2.1 nc100

nc100 は、コンパイルドライバの実行ファイルです。

nc100 は、オプションの指定によりコンパイルからリンクまでの処理を連続して行うことができます。また、nc100 の起動オプション"-as100"、"-ln100"に続けてリロケータブルマクロアセンブラ as100、リンケージエディタ ln100 のオプションを指定することができます。

1.2.2 igen100

igen100 は、インラインジェネレータの実行ファイルです。

igen100 は、cpp100 を呼び出します。

1.2.3 cpp100

cpp100 は、プリプロセッサの実行ファイルです。

cpp100 は、#で始まるマクロ(#define、#include 等)と条件コンパイル(#if~#else~#endif 等)の処理を行います。

1.2.4 ccom100

ccom100 は、コンパイラ本体の実行ファイルです。

cpp100 によって処理された C 言語ソースプログラムを as100 で処理可能なアセンブリ言語ソースプログラムに変換します。

1.2.5 aopt100

aopt100 は、アセンブラ最適マイザです。

ccom100 が出力したアセンブラコードに対して最適化を行います。

1.2.6 Call Walker & gensni

Call Walker は、プログラムの動作に必要な、スタックサイズと関数の呼び出し関係を、グラフィカルに表示するユーティリティです。また、gensni は、Call Walker で必要な情報の解析を行うユーティリティです。

Call Walker は、gensni が出力したスタック情報ファイル (.x30) 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・読み込み可能です。

Call Walker & gensni を使用するには、コンパイル時にコンパイルドライバの起動オプション-finfo を指定して、アブソリュートモジュールファイル (.x30) にインスペクタ情報が付加されるようにしてください。

1.2.7 MapViewer

MapViewer は、マップビューワの実行ファイルです。

MapViewer は、アブソリュートモジュールファイル(.x30)を処理し、リンク後のメモリ配置をグラフィカルに表示します。

MapViewer を使用するには、コンパイル時にコンパイルドライバの起動オプション"-finfo"を指定して、アブソリュートモジュールファイル(.x30)を生成してください。

1.3 注意事項

本資料に記載の製品データ、図、表に示す技術的な内容、プログラムおよ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。株式会社ルネサス エレクトロニクス及び株式会社ルネサス ソリューションズは、適用可否に対する責任は負いません。

1.3.1 コンパイラのバージョンアップ等についての注意事項

本コンパイラが生成する機械語命令(アセンブリ言語)は、コンパイル時に指定する起動オプション、バージョンアップの内容等により変化します。したがって、起動オプションの変更又はコンパイラのバージョンアップを行った場合は再度アプリケーションプログラムの動作評価を必ず行ってください。

また、割り込み処理プログラムと被割り込み処理プログラム間、リアルタイム OS 上のタスク間等で、同じ RAM データを参照し内容を変更する場合は、必ず `volatile` 指定等の排他制御を行ってください。また、ビットフィールド構造体において、メンバ名が異なっている場合においても、同一の RAM 上に確保される場合は、同様に排他制御を行ってください。

1.3.2 マイコンの機種依存部に関する注意事項

SFR 領域のレジスタへの書き込み、または読み出しには特定の命令を使用しなければならないことがあります。この特定の命令は機種ごとに異なりますので、詳しくは各マイコン機種のユーザーズマニュアルを参照してください。

本コンパイラは、SFR 領域のレジスタへの書き込み、読み出しには使用できない命令を生成する場合があります。

C 言語で SFR 領域のレジスタへの書き込み、読み出しをする場合は、`asm` 関数を使用してプログラム中に直接命令を記述してください。この場合、コンパイラのバージョン、オプションの有無に関わらず、生成されたコードに問題が無いことを必ず確認してください。

【図 1.2】のような C 言語記述を SFR 領域に行った場合、割り込み要求ビットの判定が正常に行われず意図しない動作を行う可能性があります。

```
#pragma ADDRESS TA0IC 006Ch      /* R32C/100 タイマ A0 割り込み制御レジスタ */

struct {
    char    ILVL : 3;
    char    IR : 1;          /* 割り込み要求ビット */
    char    dmy : 4;
} TA0IC;

void    wait_until_IR_is_ON(void)
{
    while (TA0IC.IR == 0)      /* 1 になるまで待つ */
    {
        ;
    }
    TA0IC.IR = 0;            /* 1 になったら 0 に戻す */
}
```

図1.2 SFR領域に対するCソースコード記述

1.4 プログラム開発例

NC100 を使用したプログラム開発例の流れを【図 1.3】に示します。このプログラムの概要を以下に示します(項目の(1)~(4)は【図 1.3】の(1)~(4)に対応します)。

- (1) C言語ソースプログラム(AA.c)をnc100でコンパイル、アセンブラas100でアセンブルし、リロケータブルオブジェクトファイル(AA.r30)を作成します。
- (2) スタートアッププログラムncrt0.a30とセクション情報を記述したインクルードファイルsect100.incを組み込むシステムに合わせて、セクションの配置/セクションサイズ/割り込みベクタテーブルの設定などを変更します。
- (3) 変更したスタートアッププログラムをアセンブルします。この結果、リロケータブルオブジェクトファイル(ncrt0.r30)を作成します。
- (4) 2つのリロケータブルオブジェクトファイル、AA.r30とncrt0.r30をnc100から実行されるリンカージェディタln100でリンクし、アプソリュートモジュールファイル(AA.x30)を作成します。

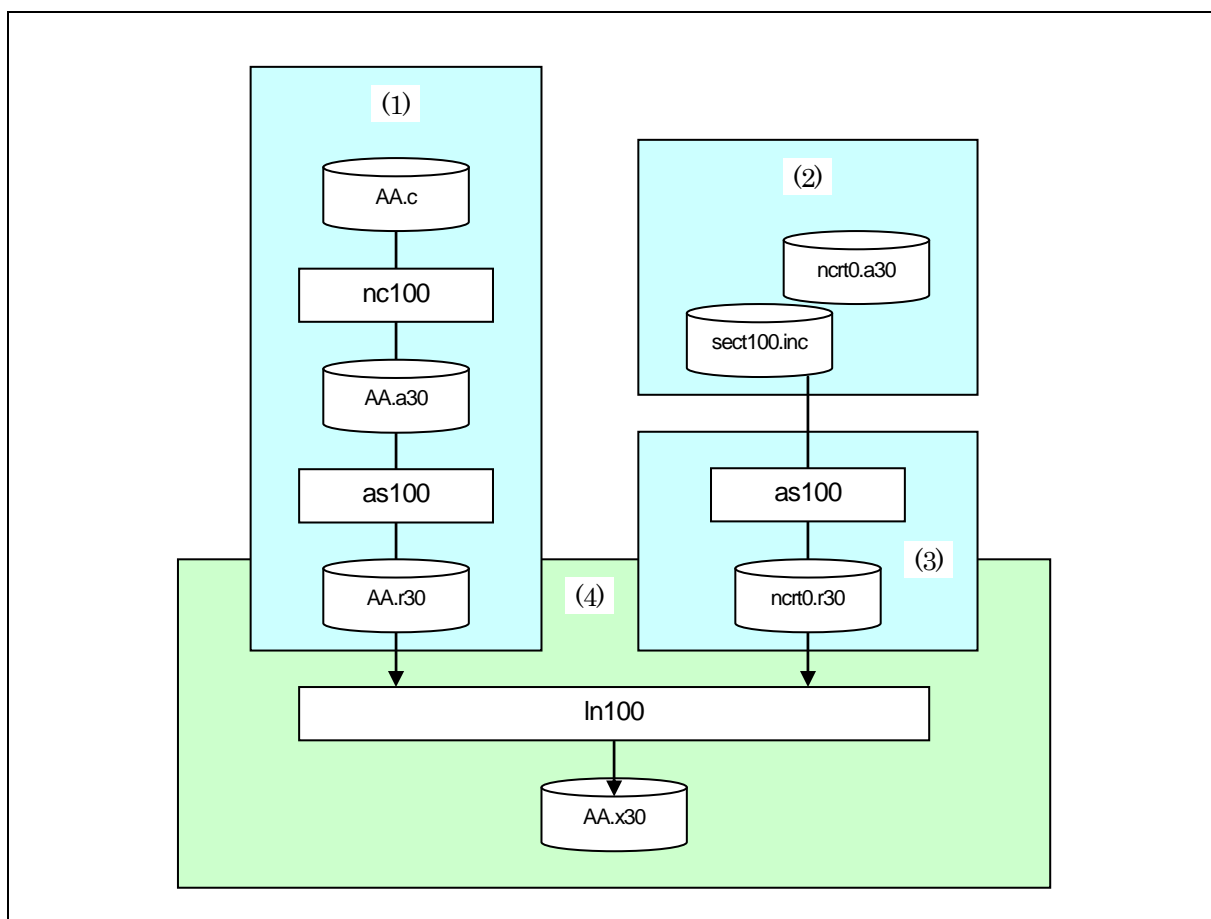


図1.3 プログラム開発フロー

【図 1.3】に示した一連の処理を記述した実行手順ファイル(makefile)の例を【図 1.4】に示します。

```
AA.x30 : ncrt0.r30 AA.r30
        nc100 -oAA ncrt0.r30 AA.r30

ncrt0.r30 : ncrt0.a30
          as100 ncrt0.a30

AA.r30 : AA.c
       nc100 -c AA.c
```

図1.4 実行手順ファイル(makefile)の記述例

また、コンパイルドライバnc100では、【図 1.4】と同様の処理をコマンドラインから【図 1.5】に示すように入力できます。

```
% nc100 -oAA ncrt0.a30 AA.c<RET>
```

% : プロンプトを示します。
<RET> : リターンキーの入力を示します。

※リンク処理を行うときは必ずスタートアッププログラムを最初に指定してください。

図1.5 nc100コマンドの入力例

1.5 NC100 の出力ファイル

サンプルプログラム `sample.c` を NC100 でコンパイルした結果、出力されるプリプロセス結果 C 言語ソースプログラム、アセンブリ言語ソースプログラムの概要を説明します。

1.5.1 出力ファイルの概要

コンパイルドライバ `nc100` は、起動オプションによって【図 1.6】に示すファイルを出力します。次項から【図 1.7】に示す C 言語ソースファイル `sample.c` をコンパイル/アセンブル/リンクした結果、出力された個々のファイル例と表示内容を説明します。

なお、`as100` 及び `ln100` が出力するリロケータブルオブジェクトファイル(拡張子 `.r30`)、プリントファイル(拡張子 `.lst`)、マップファイル(拡張子 `.map`)、等に関しては「AS100 ユーザーズマニュアル」を参照してください。

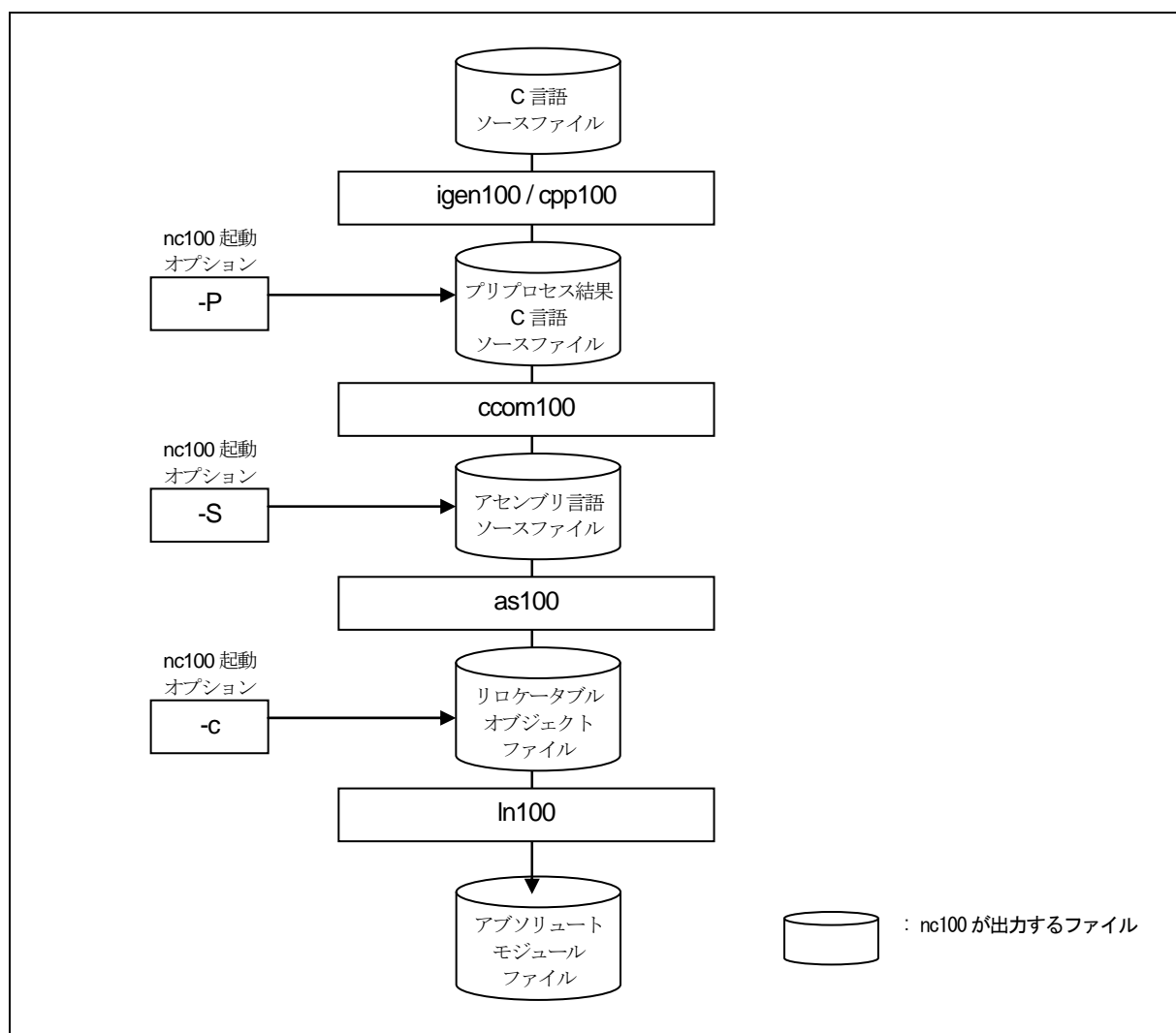


図1.6 nc100起動オプションと出力ファイルの相互関係図


```

#include <stdio.h>
#define CLR 0
#define PRN 1

void    main(void)
{
    int    flag;

    flag = CLR;
#ifdef PRN
    printf( "flag = %d¥n", flag );
#endif
}

```

図1.7 C言語ソースファイル例 (sample.c)

1.5.2 プリプロセス結果C言語ソースファイル

プリプロセッサ `cpp100` は、#で始まるプリプロセスコマンドに対して、ヘッダファイルの内容、マクロの展開、及び条件コンパイルの判定、等の処理を行います。

プリプロセス結果 C 言語ソースファイルは、`cpp100` が C 言語ソースファイルを処理した結果を格納しています。したがって、このファイルには、`#pragma`、`#line` 以外のプリプロセス行は出力されません。このファイルの内容を参照することによりコンパイラが処理を行うプログラムの内容を確認することができます。ファイルの拡張子は `i` です。

ファイルの出力例を【図 1.8】、【図 1.9】に示します。

```

typedef struct _iobuf {                                (1)
    char    _buff;
    int     _cnt;
    int     _flag;
    int     _mod;
    int     (*_func_in)(void);
    int     (*_func_out)(int);
} FILE;
:
(省略)
:
typedef long    fpos_t;
typedef unsigned long    size_t;
extern FILE    _iob[];

```

図1.8 プリプロセス結果C言語ソースファイル例 (1)

```

typedef char _far *__va_list;                                (1)

extern int      getc(FILE _far *);
extern int      getchar(void);
extern int      putc(int, FILE _far *);
extern int      putchar(int);
extern int      feof(FILE _far *);
extern int      ferror(FILE _far *);
extern int      fgetc(FILE _far *);
extern char _far *fgets(char _far *, int, FILE _far *);
extern int      fputc(int, FILE _far *);
extern int      fputs(const char _far *, FILE _far *);
:
(省略)
:
extern int      printf(const char _far *, ...);
extern int      fprintf(FILE _far *, const char _far *, ...);
extern int      sprintf(char _far *, const char _far *, ...);
:
(省略)
:
extern int      init_dev(FILE _far *, int);
extern int      speed(int, int, int, int);
extern int      init_prn(void);
extern int      _sget(void);
extern int      _sput(int);
extern int      _pput(int);
extern const char _far *_print(int(*)(), const char _far *, int _far * _far *, int _far
*);

void main(void)                                           (2)
{
    int flag;

    flag = 0 ;                                           ← (3)

    printf( "flag = %d\n", flag );                       ← (4)
}

```

図1.9 プリプロセス結果C言語ソースファイル例 (2)

プリプロセス結果C言語ソースファイルの内容を以下に説明します。項目番号の(1)~(4)は【図1.8】、【図1.9】中の(1)~(4)にそれぞれ対応しています。

- (1) #includeで指定したヘッダファイルstdio.hの展開部を示します。
- (2) マクロを展開した結果のC言語ソースプログラムを示します。
- (3) #defineで指定されたCLRを0として展開していることを示します。
- (4) #define で指定されたPRN が1であるためコンパイル条件が有効となり、printf 関数が出力されていることを示します。

1.5.3 アセンブリ言語ソースファイル

このファイルは、コンパイラ `ccom100` がプリプロセス結果 C 言語ソースファイルを `AS100` で処理可能なアセンブリ言語に変換したファイルです。ここで出力されるファイルは、拡張子 `.a30` で示されるアセンブリ言語ソースファイルです。

ファイルの出力例を【図 1.10】、【図 1.11】に示します。なお、アセンブリ言語ソースファイルの出力には、`nc100` の起動オプション `"-dsource(-dS)"` を指定して行単位で C 言語ソースファイルの内容をコメントとして表示させています。

```

        ._LANG    'C', 'X.XX.XX.XXX', 'REV.X'

;##   C Compiler          OUTPUT
;##   ccom100 Version X.XX.XX.XXX
;##   Copyright(C) XXXX. Renesas Electronics Corp.
;##   and Renesas Solutions Corp., All Rights Reserved.
;##   Compile Start Time XXX XX XX XX:XX:XX XXXX

;##   COMMAND_LINE: ccom100 -dS -o sample.a30 sample.i

-----
;##   Normal Optimize          OFF          (1)
;##   ROM size Optimize       OFF
;##   Speed Optimize          OFF
;##   Default ROM is          far
;##   Default RAM is          near
-----

        .GLB      __SB__
        .SB       __SB__
        .FB       0

;## #   FUNCTION main
;## #   ARG Size(4)          Auto Size(0)          Context Size(4)

        .SECTION program, CODE, ALIGN
        ._file    'sample.c'
        .align
        ._line    6
;## #   C_SRC :          {
        .glb      _main
_main:
        ._line    9
;## #   C_SRC :          flag = CLR;
        mov.l     #00000000H, R2R0 ; flag
        ._line    11
;## #   C_SRC :          printf("flag = %d\n", flag);          ← (2)
        push.l   R2R0 ; flag
        push.l   #__T0
        jsr      _printf
        add.l   #08H, SP
        ._line    13
;## #   C_SRC :          }
        rts
E1:
        :
        (省略)
        :
        .glb     __iob
        .glb     $getc
        .glb     _getchar
        .glb     $putc
        .glb     $putchar
        .glb     $feof
        .glb     $ferror
        .glb     $fgetc
        .glb     $fgets
        .glb     $putc
        :
        (省略)
        :

```

図1.10 アセンブリ言語ソースファイル例 (1) (sample.a30)

```
      .SECTION rom_FAR,ROMDATA,ALIGN
___T0:
      .byte  66H      ; 'f'
      .byte  6cH      ; 'l'
      .byte  61H      ; 'a'
      .byte  67H      ; 'g'
      .byte  20H      ; ' '
      .byte  3dH      ; '='
      .byte  20H      ; ' '
      .byte  25H      ; '%'
      .byte  64H      ; 'd'
      .byte  0aH
      .byte  00H
      .END

;## Compile End Time XX XXX XX XX:XX:XX XXXX
```

図1.11 アセンブリ言語ソースファイル例 (2) (sample.a30)

アセンブリ言語ソースファイルの内容を以下に説明します。項目番号の(1)~(2)は【図 1.10】中の(1)~(2)に対応しています。

- (1) 最適化オプションの状態とROM及びRAMに対するnear/far属性の初期設定の情報を示しています。
- (2) nc100の起動オプション"-dsource(-dS)"を指定したときにC言語ソースファイルの内容がコメントで表示されます。

第2章 コンパイラの基本的な使い方

この章では、コンパイルドライバの起動方法と起動オプションの機能を説明します。起動オプションの説明では、コンパイルドライバから起動できるアセンブラとリンカージェディタの起動オプションを併せて記載しています。

2.1 コンパイラの起動

2.1.1 コンパイルドライバのコマンドの入力書式

コンパイルドライバは、コンパイラの各コマンドとアセンブルコマンド及びリンクコマンドを起動し、機械語データファイルを生成します。このコンパイルドライバを起動するためには、以下の情報(入力パラメータ)が必要となります。

- (1) C言語ソースファイル
- (2) アセンブリ言語ソースファイル
- (3) リロケータブルオブジェクトファイル
- (4) 起動オプション(必要に応じて記述する項目)

これらの項目をコマンド行に入力します。項目(1)、(2)、(3)、のいずれか一つは最低限、入力してください。

【図 2.1】に入力書式を、【図 2.2】に入力例を示します。

入力例では、

- (1) スタートアッププログラムncrt0.a30をアセンブル
- (2) C言語ソースプログラムsample.cをコンパイル/アセンブル
- (3) リロケータブルオブジェクトファイルncrt0.r30とsample.r30をリンク

を行い、アブソリュートモジュールファイル sample.x30 を作成するときの記述例を示します。起動オプションには、

- アブソリュートモジュールファイル名sample.x30の指定-oオプション
- アセンブル時のリストファイル(拡張子.lst)の出力指定.....-as100 "-l"オプション
- リンク時のマップファイル(拡張子.map)の出力指定-ln100 "-ms"オプション

を行っています。

```
% nc100△[起動オプション]△<[アセンブリ言語ソースファイル名]△
      [リロケータブルオブジェクトファイル名]△[C言語ソースファイル名]>
```

% : プロンプトを示します。

< > : 必須項目を示します。

[] : 必要に応じて記述する項目を示します。

△ : スペースを示します。

図2.1 コンパイルドライバコマンドの入力書式

```
% nc100 -osample -as100 "-l" -ln100 "-ms" ncr0.a30 sample.c<RET>
```

<RET> : リターンキーの入力を示します。

※リンク時には必ずスタートアッププログラムを先に指定してください。

図2.2 コンパイルドライバコマンドの入力例

2.1.2 コマンドファイル

コンパイルドライバは、複数のコマンドオプションを記述したファイル(コマンドファイル)を読み込んでコンパイル処理を行うことができます。

コマンドファイルを使用することにより、Microsoft Windows(TM)等のコマンド行の文字数制限を回避することができます。

(a) コマンドファイルの入力書式

```
% nc100△[起動オプション]△<@ファイル名>△[起動オプション]
```

% : プロンプトを示します。

< > : 必須項目を示します。

[] : 必要に応じて記述する項目を示します。

△ : スペースを示します。

図2.3 コマンドファイルの入力書式

```
% nc100 -c @test.cmd -g<RET>
```

<RET> : リターンキーの入力を示します。

図2.4 コマンドファイルの入力例

コマンドファイルの記述は以下のようになります。

test.cmd の記述	→	<pre>ncr0.a30<CR> sample1.c sample2.r30<CR> -g -as100 -l<CR> -o<CR> sample<CR></pre>
--------------	---	--

<CR> : 改行を示します。

図2.5 コマンドファイルの記述例

(b) コマンドファイルの記述規定

コマンドファイルの記述には以下の規定があります。

- 一度に指定できるコマンドファイルは、1 ファイルのみです。同時に複数のコマンドファイルを指定することはできません。
- コマンドファイル内にコマンドファイルを指定できません。
- コマンドファイル内には、コマンド行を複数行にわたって記述できます。
- コマンドファイル内の改行は、空白文字に置き換えられます。
- コマンドファイルの一行に記述可能な文字数は、2048文字までです。2048文字を越えた場合はエラーとなります。

(c) コマンドファイル使用時の注意事項

コマンドファイル名にディレクトリパスを指定できます。指定したディレクトリパスにファイルが存在しない場合はエラーとなります。

リンク時にファイルを指定するために、拡張子".cm\$"のln100用コマンドファイルを自動生成します。このため、拡張子が".cm\$"のファイルがある場合は、上書きされる可能性があります。拡張子が".cm\$"のファイルは、使用しないでください。

同時に2ファイル以上のコマンドファイルは指定できません。複数ファイルを指定した場合は、"Too many command files."のエラーメッセージを表示し終了します。

2.1.3 起動オプションに関する注意事項**(a) 起動オプションの記述に関する注意事項**

コンパイルドライバの起動時オプションは、アルファベットの大文字と小文字を区別します。誤って入力した場合、そのオプションによる機能は取り消されます。

(b) コンパイルドライバの制御に関するオプションの優先順位

コンパイルドライバの制御に関するオプションには、以下の優先順位があります。

-E	-P	-S	-c
← 高	優先順位	低	→

例えば、

- "-c" : リロケータブルファイル(拡張子.r30)を作成して処理を終える
- "-S" : アセンブリ言語ソースファイル(拡張子.a30)を作成して処理を終える

を同時に指定した場合は"-S"オプションが優先されます。つまり、コンパイルドライバは、アセンブラ以後の処理を行いません。この場合は、アセンブリ言語ソースファイルのみが生成されます。

リロケータブルファイルを作成し、かつ、アセンブリ言語ソースファイルも同時に作成したい場合は、"-dsource (短縮形 -dS)"を使用してください。

2.1.4 nc100 の起動オプション

(a) コンパイルドライバの制御に関するオプション

【表 2.1】にコンパイルドライバの制御に関する起動オプションを示します。

表2.1 コンパイルドライバ制御オプション

オプション	機能
-c	リロケータブルファイル(拡張子.r30)を作成し、処理を終了します。 ¹
-D 識別子名	識別子を定義します。#define と同じ機能です。
-dsource (短縮形 -dS)	C 言語ソースリストをコメントとして出力したアセンブリ言語ソースファイル(拡張子".a30")を生成します(アセンブル後も削除しません)。
-dsource_in_list (短縮形 -dSL)	"-dsource(-dS)"の機能に加えて、アセンブリ言語リストファイル(.lst)を生成します。
-E	プリプロセスコマンドのみを処理し結果を標準出力に出力します。
-I ディレクトリ名	プリプロセスコマンドの#include で参照するファイルを検索するディレクトリ名を指定します。ディレクトリは最大 256 個まで指定可能です。
-P	プリプロセスコマンドのみを処理し、ファイル(拡張子.i)を作成します。
-S	アセンブリ言語ソースファイル(拡張子.a30)を作成し、処理を終了します。
-silent	起動時のコピーライトメッセージを出力しません。
-U プリデファインドマクロ名	指定したプリデファインドマクロを未定義にします。

(b) 出力ファイル指定オプション

【表 2.2】に出力するアブソリュートモジュールファイルの名称を指定する起動オプションを示します。

表2.2 出力ファイル指定オプション

オプション	機能
-dir ディレクトリ名	ln100 が生成するファイル(アブソリュートモジュールファイル、マップファイル、等)の出力先ディレクトリを指定できます。
-o ファイル名	ln100 が生成するファイル(アブソリュートモジュールファイル、マップファイル、等)の名称を指定します。また、ディレクトリ名を含んだパス名も指定できます。ファイルの拡張子は必ず省略してください。

(c) バージョン及びコマンドライン情報表示オプション

【表 2.3】に使用するクロスツールのバージョン及びコマンドラインを表示する起動オプションを示します。

表2.3 バージョン情報及びコマンドライン表示オプション

オプション	機能
-v	実行中のコマンドプログラム名及びコマンドラインを表示します。
-V	コンパイラの各プログラムの起動時メッセージを表示し、処理を終了します(コンパイル処理は行いません)。

¹ 起動オプション-c、-E、-P、及び-Sを指定しない場合、nc100はln100まで制御を行い、アブソリュートモジュールファイル(拡張子.x30)まで作成します。

(d) デバッグ用オプション

【表 2.4】に C 言語レベルデバッグ情報を出力するデバッグの起動オプションを示します。

表2.4 デバッグ用オプション

オプション	機能
-g	デバッグ情報をアセンブリ言語ソースファイル(拡張子.a30)に出力します。これにより、C 言語レベルデバッグが可能になります。
-genter	関数呼び出し時に必ず enter 命令を出力します。 デバッグのスタックトレース機能を使用するときは、必ずこのオプションを指定してください。

(e) 最適化オプション

【表 2.5】にプログラムの実行速度及びROM容量を最小にする最適化を行う起動オプションを示します。

表2.5 最適化オプション

オプション	短縮形	機能
-O[1~5]	なし	レベル毎に速度及びROM容量ともに効果がある最適化を行います。
-O5OA	なし	最適化オプション"-O5"選択時におけるビット操作命令を使用したコード生成を抑制します。
-OR	なし	ROM容量を重視した最大限の最適化を行います。
-OR_MAX	-ORM	ROMサイズに効果のある最適化を最大限に行います。
-OS	なし	速度を重視した最大限の最適化を行います。
-OS_MAX	-OSM	速度に効果のある最適化を最大限に行います。
-Ocompare_byte_to_word	-OCBTW	連続した領域のバイト単位の比較をワード単位で行います。
-Oconst	-OC	const修飾子で宣言した外部変数の参照を定数に置き換える最適化を行います。なお、定数領域は確保されます。
-Ofile_inline	-OFI	全てのインライン関数に対して、インライン展開を行います。
-Oinline_line	-OIL	インライン展開する関数のサイズ(行数)を変更します。
-Oglob_jump	-OGJ	外部分岐の最適化を行います。
-Oglobal_to_inline	-OGTI	グローバル関数を、inline宣言扱いにします。
-Oloop_unroll[=ループ回数]	-OLU	ループ文を回さずに、ループ回数分コードを展開します。"ループ回数"は省略可能、省略時は最大5回のループ文が対象となります。
-Ono_bit	-ONB	ビット操作をまとめる最適化を抑制します。
-Ono_break_source_debug	-ONBSD	ソース行情報に影響する最適化を抑制します。
-Ono_float_const_fold	-ONFCF	浮動小数点の定数畳み込み処理を抑制します。
-Ono_logical_or_combine	-ONLOC	論理ORをまとめる最適化を抑制します。
-Ono_asmopt	-ONA	アセンブラオプティマイザ"aopt100"による最適化を抑制します。
-Osp_adjust	-OSA	スタック補正コードを取り除く最適化を行います。 これによりROM容量を削減することができます。 ただし、使用するスタック量が多くなる可能性があります。
-Ostatic_to_inline	-OSTI	static宣言された関数を、inline宣言扱いにします。

(f) 生成コード変更オプション

【表 2.6】に本コンパイラが生成するアセンブリ言語を制御する起動オプションを示します。

表2.6 生成コード変更オプション

オプション	短縮形	機能
-fansi	なし	"-fnot_reserve_far_and_near"、"-fnot_reserve_asm"、"-fnot_reserve_inline"及び"-fextend_to_int"を有効にします。
-fconst_not_ROM	-fCNR	const で指定した型を ROM データとして扱いません。
-fdouble_32	-fD32	double 型を float 型として処理します。
-fenable_register	-fER	レジスタ記憶クラスを有効にします。
-fextend_to_int	-fETI	char型およびshort型データをint型に拡張し演算します(ANSI規格で定められた拡張を行います)。 ²
-ffar_RAM	-fFRAM	RAM データのデフォルト属性を far にします。
-finfo	なし	"Call Walker"、"Map Viewer"に必要な情報を出力します。
-fint_16	-fI16	int 型を 16 ビット幅で扱います。
-fJSRW	なし	関数呼び出しの命令のデフォルトを JSR.W 命令に変更します。
-fnear_ROM	-fNROM	ROM データのデフォルト属性を near にします。
-fno_align	-fNA	関数の先頭アドレスのアライメントを行いません。
-fno_switch_table	-fNST	switch 文に対し、比較を行ってから分岐するコードを生成します。
-fnot_address_volatile	-fNAV	#pragma ADDRESS(#pragma EQU)で指定した変数を volatile で指定した変数とみなしません。
-fnot_reserve_asm	-fNRA	asm を予約語にしません("_asm"のみ有効になります)。
-fnot_reserve_far_and_near	-fNRFAN	far、near を予約語にしません(_far、_near のみ有効になります)。
-fnot_reserve_inline	-fNRI	inline を予約語にしません(_inline のみ予約語となります)。
-fsigned_char	-fSC	符号指定のない char 型を signed char 型として扱います。
-fswitch_other_section	-fSOS	switch 文に対するテーブルジャンプをプログラムセクションとは別セクションに出力します。
-fuse_FPU	-fUF	FPU 命令を出力します。

(g) ライブラリ指定オプション

【表 2.7】にライブラリファイルを指定する起動オプションを示します。

表2.7 ライブラリ指定オプション

オプション	機能
-lライブラリファイル名	リンク時に使用するライブラリを指定します。

² ANSI 規格では、(unsigned)char型、signed char型、short型およびunsigned short型データを評価する時に必ずint 型に拡張します。

これは、「c1 = c2 * 2 / c3;」のようなchar型の演算を行う場合に、演算の途中でchar型をオーバーフローし、結果が予期せぬ値になるのを防ぐためです。

(h) 警告オプション

【表 2.8】に本コンパイラの言語仕様に関する記述の間違いに対して警告(ワーニングメッセージ)を出力する起動オプションを示します。

表2.8 警告オプション

オプション	短縮形	機能
-Wall	なし	検出可能な警告("-Wlarge_to_small"、"Wno_used_argument" で出力される警告を除く)をすべて表示します。
-Wccom_max_warnings = ワーニング回数	-WCMW	ccom100 の出力するワーニングの回数の上限を指定できます。
-Werror_file<ファイル名>	-WEF	タグファイルを出力します。
-Wignore_near_pointer	-WINP	near ポインタを far ポインタとして扱う際の警告を抑制します。
-Wlarge_to_small	-WLTS	大きいサイズから小さいサイズへの暗黙の代入に対して、ワーニングを出力します。
-Wmake_tagfile	-WMT	error および warning が発生した場合はファイル毎にタグファイルを出力します。
-Wmultiple_tentative_definitions	-WMTD	同一変数名に対して複数の仮定義がある場合に警告します。
-Wnesting_comment	-WNC	コメント中に"/"を記述した場合に警告を出します。
-Wno_stop	-WNS	エラーが発生してもコンパイル作業を停止しません。
-Wno_used_argument	-WNUA	引数を持つ関数を定義した場合に、使用していない引数に対してワーニングを出力します。
-Wno_used_function	-WNUF	リンク時に未使用のグローバル関数を表示します。
-Wno_used_static_function	-WNUSF	コード生成が不要な static 関数名を表示します。
-Wno_warning_stdlib	-WNWS	"-Wnon_prototype"指定時や"-Wall"指定時に本オプションを指定すると、「プロトタイプ宣言されていない標準ライブラリに対する警告」を抑制します。
-Wnon_prototype	-WNP	プロトタイプ宣言されていない関数を使用した場合、警告を出します。
-Wstdout	なし	エラーメッセージをホストマシンの標準出力(stdout)に出力します。
-Wstop_at_link	-WSAL	リンク時にワーニングが発生した場合、アブソリュートモジュールファイルの生成を抑制します。
-Wstop_at_warning	-WSAW	ワーニング発生時にコンパイル処理を停止します。
-Wundefined_macro	-WUM	#if の中で未定義のマクロを使用した場合に警告します。
-Wuninitialize_variable	-WUV	初期化されていない auto 変数に対してワーニングを出力します。
-Wunknown_pragma	-WUP	サポートしていない#pragma を使用した場合、警告を出します。

(i) アセンブル / リンクオプション

【表 2.9】 にas100 及びln100 のオプションを指定する起動オプションを示します。

表2.9 アセンブル/リンクオプション

オプション	機能
-as100△< オプション>	アセンブルコマンド as100 のオプションを指定します。2 個以上のオプションを渡す場合は、" (ダブルクォーテーション) で囲んでください。
-ln100△< オプション>	リンクコマンド ln100 オプションを指定します。2 個以上のオプションを渡す場合は、" (ダブルクォーテーション) で囲んでください。

2.2 スタートアッププログラムの準備

C 言語で記述したプログラムを ROM 化するために、本コンパイラでは、マイコンの初期設定、セクションの配置、割り込みベクタアドレステーブル、等を設定するアセンブリ言語で記述したサンプルのスタートアッププログラムを製品に付属しています。スタートアッププログラムを組み込むシステムに合わせて変更する必要があります。

ここでは、スタートアッププログラムについてと、そのカスタマイズの仕方について説明します。

2.2.1 スタートアッププログラムのサンプル

スタートアッププログラムは、以下の2つのファイルで構成しています。

- `nrcrt0.a30`
リセット直後に実行されるプログラムを記述します。
- `sect100.inc`
このファイルは、`nrcrt0.a30`からインクルードされ、セクションの配置(メモリの配置)を定義します。

`nrcrt0.a30` のソースプログラムリストを【図 2.6】から【図 2.11】に、`sect100.inc`のソースプログラムリストを【図 2.12】から【図 2.17】、にそれぞれ示します。

```

;*****;
;
; C COMPILER for R32C/100
; Copyright(C) XXXX. Renesas Electronics Corp.
; and Renesas Solutions Corp., All rights reserved.
;
; nrcrt0.a30 : startup program
;
; This program is applicable when using the basic I/O library
;
; $Id: nrcrt0.a30,v X.XX XXXX/XX/XX XX:XX:XX XXXXX Exp $
;
;*****;

;-----
; HEAP SIZE definition                                     ← (1)
;-----
.if __HEAP__ == 1                                     ; for HEW
HEAPSIZE .equ    0h

.else
.if __HEAPSIZE__ == 0
HEAPSIZE .equ    300h

.else                                     ; for HEW
HEAPSIZE .equ    __HEAPSIZE__

.endif
.endif;

```

(1) 使用する heap サイズを定義します。

図2.6 スタートアッププログラムnrcrt0.a30リスト (1)

```

;-----
; STACK SIZE definition                               ← (2)
;-----
.if __USTACKSIZE__ == 0

STACKSIZE      .equ      300h

.else
                ; for HEW

STACKSIZE      .equ      __USTACKSIZE__

.endif

;-----
; INTERRUPT STACK SIZE definition                     ← (3)
;-----
.if __ISTACKSIZE__ == 0

ISTACKSIZE     .equ      300h

.else
                ; for HEW

ISTACKSIZE     .equ      __ISTACKSIZE__

.endif

;-----
; INTERRUPT VECTOR ADDRESS definition                ← (4)
;-----
VECTOR_ADR     .equ      0FFFFFFBDCh

;-----
; Section allocation
;-----
        .list OFF
        .include sect100.inc                         ← (5)
        .list ON

```

- (2) ユーザースタックサイズを定義します。
- (3) 割り込みスタックサイズを定義します。
- (4) 割り込みベクタテーブルの開始アドレスを定義します。
- (5) sect100.inc をインクルードします。

図2.7 スタートアッププログラムμnct0.a30リスト (2)

```
-----;
; SB AREA DEFINITION          ;
-----;
      .glob          __SB__
__SB__ .equ          data_SB8_top
-----;
; INITIALIZE MACRO DEFINITION ;
-----;
BZERO      .macro          TOP_,SECT_
            mov.b          #00H,R0L
            mov.l          #TOP_,A1
            mov.l          #sizeof SECT_,R7R5
            sstr.b
            .endm
BCOPY      .macro          FROM_,TO_,SECT_
            mov.l          #FROM_,A0
            mov.l          #TO_,A1
            mov.l          #sizeof SECT_,R7R5
            smovf.b
            .endm
```

図2.8 スタートアッププログラム△nct0.a30リスト (3)


```

;-----;
; INTERRUPT SECTION ;
;-----;
        .insf          start, S, 0
        .glb          start
        .section      interrupt, code, align
start:                                     ← (6)
;-----;
; after reset, this program will start ;
;-----;
ldc          #istack_top,ISP          ; istack pointer
ldc          #0080H,FLG              ; switch to usp ← (7)
ldc          #stack_top,SP          ; stack pointer
ldc          #data_SB8_top,SB       ; sb register
fset        b
ldc          #data_SB8_top,SB       ; bsb register
fclr        b
ldc          #VECTOR_ADR,INTB       ; vector address
;-----;
; zero clear BSS ;
;-----;
BZERO       bss_SB8_top, bss_SB8
;
BZERO       bss_SB16_top, bss_SB16
BZERO       bss_NEAR_top, bss_NEAR
BZERO       bss_FAR_top, bss_FAR
BZERO       bss_EXT_top, bss_EXT
BZERO       bss_MON1_top, bss_MON1
BZERO       bss_MON2_top, bss_MON2
BZERO       bss_MON3_top, bss_MON3
BZERO       bss_MON4_top, bss_MON4
;-----;
; initialize DATA ;
;-----;
BCOPY       data_SB8_INIT_top, data_SB8_top, data_SB8
;
BCOPY       data_SB16_INIT_top, data_SB16_top, data_SB16
BCOPY       data_NEAR_INIT_top, data_NEAR_top, data_NEAR
BCOPY       data_FAR_INIT_top, data_FAR_top, data_FAR
BCOPY       data_EXT_INIT_top, data_EXT_top, data_EXT
BCOPY       data_MON1_INIT_top, data_MON1_top, data_MON1
BCOPY       data_MON2_INIT_top, data_MON2_top, data_MON2
BCOPY       data_MON3_INIT_top, data_MON3_top, data_MON3
BCOPY       data_MON4_INIT_top, data_MON4_top, data_MON4

```

- (6) リセット直後はこのラベル start からスタートします。
(7) 割り込み許可レベル及び各種フラグの設定を行います。
(8) bss セクションのゼロクリア処理を行います。
(9) data セクションの初期値を RAM 領域に転送します。

図2.9 スタートアッププログラムnrcr0.a30リスト (4)

```

;-----;
; initialize heap manager ; ← (10)
;-----;
.if __HEAP__ != 1
.glb      __mnext
.glb      __msize
mov.l    #heap_top,__mnext
mov.l    #HEAPSIZE,__msize
.endif

;-----;
; initialize standard I/O ; ← (11)
;-----;
.if __STANDARD_IO__ == 1
.glb      __init
.call    __init,G
jsr.a   __init
.endif

;-----;
; invoke main() function ; ← (12)
;-----;
ldc     #0H,FB ; for DEBUGGER
.glb    _main
jsr.a   _main

```

(10) heap 領域の初期化を行います。メモリ管理関数を使用しない場合にはコメントにします。

(11) 標準入出力の初期化を行う `_init` 関数を呼び出します。入出力関数を使用しない場合にはコメントにします。

(12) `main` 関数の呼び出しを行います。

※ `main` 関数呼び出し時には、割り込みは禁止になっています。
 割り込みを使用する際には "FSET" 命令により割り込みを許可してください。

図2.10 スタートアッププログラム `ncrt0.a30` リスト (5)

```

;=====
;-----;
; exit() function ; ← (13)
;-----;
.glb      _exit
.glb      $exit
_exit:
$exit:
; End of execution

jmp      _exit
.einsf
;-----;
; dummy interrupt function ; ← (14)
;-----;
.glb      dummy_int

dummy_int:
reit
.end
;*****;
; ;
; End of R32C/100 start up ;
; ;
;*****;

```

(13) exit 関数部です。

(14) ダミーの割り込み処理関数です。

図2.11 スタートアッププログラムncrt0.a30リスト (6)


```

;-----;
; HEAP SECTION                                     ;
;-----;
heap_top:      .section   heap,   data, align
               .blkb     HEAPSIZE

;-----;
; SB RELATIVE RAM SECTIONS                         ;
;-----;
;           .section   data_SB8, data
;           .org       00008000H
;data_SB8_top:
;           .section   bss_SB8, data, align
; bss_SB8_top:
;           .section   data_SB16, data, align
;data_SB16_top:
;           .section   bss_SB16, data, align
; bss_SB16_top:

;-----;
; FAR RAM SECTIONS                               ;
;-----;
data_FAR_top:  .section   data_FAR, data, align
bss_FAR_top:   .section   bss_FAR, data, align

;-----;
; EXTENDED RAM SECTIONS                         ;
;-----;
;           .section   data_EXT, data
;           .org       00800000H
data_EXT_top:  .section   bss_EXT, data, align
bss_EXT_top:

;-----;
; EXTENDED ROM SECTIONS                       ;
;-----;
;           .section   data_EXT_INIT,   romdata
;           .org       0FF00000H
data_EXT_INIT_top:
;           .section   rom_EXT,   romdata, align
rom_EXT_top:
;           .section   program_EXT,   code, align

;-----;
; FAR ROM SECTIONS                             ;
;-----;
;           .section   rom_FAR,   romdata
;           .org       0FFE0000H
rom_FAR_top:

```

図2.13 スタートアッププログラムsect100.incリスト (8)

```
-----;
; INITIAL DATA SECTIONS
;-----;
        .section data_NEAR_INIT, romdata
        .org      0FFFF0000H
data_NEAR_INIT_top:
        .section data_MON1_INIT, romdata, align
data_MON1_INIT_top:
        .section data_MON2_INIT, romdata, align
data_MON2_INIT_top:
        .section data_MON3_INIT, romdata, align
data_MON3_INIT_top:
        .section data_MON4_INIT, romdata, align
data_MON4_INIT_top:
        .section data_SB8_INIT, romdata, align
data_SB8_INIT_top:
;        .section data_SB16_INIT, romdata, align
;data_SB16_INIT_top:
        .section data_FAR_INIT, romdata, align
data_FAR_INIT_top:

;-----;
; SWITCH TABLE SECTIONS
;-----;
        .section switch_table, romdata, align

;-----;
; CODE SECTIONS
;-----;
        .section program, code, align

        .section interrupt, code, align

;-----;
; NEAR ROM SECTIONS
;-----;
;        .section rom_NEAR, romdata
;        .org      0FFFF8000H
;rom_NEAR_top:
```

図2.14 スタートアッププログラムsect100.incリスト (9)

```

;-----;
; VARIABLE VECTOR SECTION          ;
;-----;
                .section vector, romdata
                .org      VECTOR_ADR
.if      __MVT__ == 1
.lword   dummy_int      ; BRK                (software int 0)
.lword   dummy_int      ; reservation area    (software int 1)
.lword   dummy_int      ; uart5 trance/NACK    (software int 2)
.lword   dummy_int      ; uart5 receive/ACK    (software int 3)
.lword   dummy_int      ; uart6 trance/NACK    (software int 4)
.lword   dummy_int      ; uart6 receive/ACK    (software int 5)
.lword   dummy_int      ; uart5/uart6 bus collision (software int 6)
.lword   dummy_int      ; reservation area    (software int 7)
.lword   dummy_int      ; DMA0                 (software int 8)
.lword   dummy_int      ; DMA1                 (software int 9)
.lword   dummy_int      ; DMA2                 (software int 10)
.lword   dummy_int      ; DMA3                 (software int 11)
.lword   dummy_int      ; TIMER A0             (software int 12)
.lword   dummy_int      ; TIMER A1             (software int 13)
.lword   dummy_int      ; TIMER A2             (software int 14)
.lword   dummy_int      ; TIMER A3             (software int 15)
.lword   dummy_int      ; TIMER A4             (software int 16)
.lword   dummy_int      ; uart0 trance/NACK    (software int 17)
.lword   dummy_int      ; uart0 receive/ACK    (software int 18)
.lword   dummy_int      ; uart1 trance/NACK    (software int 19)
.lword   dummy_int      ; uart1 receive/ACK    (software int 20)
.lword   dummy_int      ; TIMER B0             (software int 21)
.lword   dummy_int      ; TIMER B1             (software int 22)
.lword   dummy_int      ; TIMER B2             (software int 23)
.lword   dummy_int      ; TIMER B3             (software int 24)
.lword   dummy_int      ; TIMER B4             (software int 25)
.lword   dummy_int      ; INT5                 (software int 26)
.lword   dummy_int      ; INT4                 (software int 27)
.lword   dummy_int      ; INT3                 (software int 28)
.lword   dummy_int      ; INT2                 (software int 29)
.lword   dummy_int      ; INT1                 (software int 30)
.lword   dummy_int      ; INT0                 (software int 31)
.lword   dummy_int      ; TIMER B5             (software int 32)
.lword   dummy_int      ; uart2 trance/NACK    (software int 33)
.lword   dummy_int      ; uart2 receive/ACK    (software int 34)
.lword   dummy_int      ; uart3 trance/NACK    (software int 35)
.lword   dummy_int      ; uart3 receive/ACK    (software int 36)
.lword   dummy_int      ; uart4 trance/NACK    (software int 37)
.lword   dummy_int      ; uart4 receive/ACK    (software int 38)
.lword   dummy_int      ; uart2 bus collision  (software int 39)
.lword   dummy_int      ; uart3/uart0 bus collision (software int 40)
.lword   dummy_int      ; uart4/uart1 bus collision (software int 41)
.lword   dummy_int      ; A-D Convert         (software int 42)
.lword   dummy_int      ; input key           (software int 43)
.lword   dummy_int      ; intelligent I/O 0   (software int 44)
.lword   dummy_int      ; intelligent I/O 1   (software int 45)
.lword   dummy_int      ; intelligent I/O 2   (software int 46)
.lword   dummy_int      ; intelligent I/O 3   (software int 47)
.lword   dummy_int      ; intelligent I/O 4   (software int 48)
.lword   dummy_int      ; intelligent I/O 5   (software int 49)
.lword   dummy_int      ; intelligent I/O 6   (software int 50)
.lword   dummy_int      ; intelligent I/O 7   (software int 51)
.lword   dummy_int      ; intelligent I/O 8   (software int 52)
.lword   dummy_int      ; intelligent I/O 9   (software int 53)
.lword   dummy_int      ; intelligent I/O 10  (software int 54)

```

図2.15 スタートアッププログラムsect100.incリスト (10)

```

.word dummy_int ; intelligent I/O 11 (software int 55)
.word dummy_int ; reservation area (software int 56)
.word dummy_int ; reservation area (software int 57)
.word dummy_int ; reservation area (software int 58)
.word dummy_int ; CAN1WU (software int 59)
.word dummy_int ; reservation area (software int 60)
.word dummy_int ; reservation area (software int 61)
.word dummy_int ; reservation area (software int 62)
.word dummy_int ; reservation area (software int 63)
.word dummy_int ; reservation area (software int 64)
.word dummy_int ; reservation area (software int 65)
.word dummy_int ; reservation area (software int 66)
.word dummy_int ; reservation area (software int 67)
.word dummy_int ; Audio interface 0 (software int 68)
.word dummy_int ; Sound field processor (software int 69)
.word dummy_int ; reservation area (software int 70)
.word dummy_int ; reservation area (software int 71)
.word dummy_int ; reservation area (software int 72)
.word dummy_int ; reservation area (software int 73)
:
(省略)
:
.word dummy_int ; reservation area (software int 89)
.word dummy_int ; reservation area (software int 90)
.word dummy_int ; reservation area (software int 91)
.word dummy_int ; reservation area (software int 92)
.word dummy_int ; INT8 (software int 93)
.word dummy_int ; INT7 (software int 94)
.word dummy_int ; INT6 (software int 95)
.word dummy_int ; CAN0 trance (software int 96)
.word dummy_int ; CAN0 receive (software int 97)
.word dummy_int ; CAN0 error (software int 98)
.word dummy_int ; CAN1 trance (software int 99)
.word dummy_int ; CAN1 receive (software int 100)
.word dummy_int ; CAN1 error (software int 101)
.word dummy_int ; reservation area (software int 102)
.word dummy_int ; reservation area (software int 103)
.word dummy_int ; reservation area (software int 104)
.word dummy_int ; reservation area (software int 105)
:
(省略)
:
.word dummy_int ; reservation area (software int 120)
.word dummy_int ; reservation area (software int 121)
.word dummy_int ; reservation area (software int 122)
.word dummy_int ; reservation area (software int 123)
.word dummy_int ; uart7 trance (software int 124)
.word dummy_int ; uart7 receive (software int 125)
.word dummy_int ; uart8 trance (software int 126)
.word dummy_int ; uart8 receive (software int 127)
.word dummy_int ; software int 128
.word dummy_int ; software int 129
.word dummy_int ; software int 130
.word dummy_int ; software int 131
.word dummy_int ; software int 132

```

図2.16 スタートアッププログラムsect100.incリスト (11)


```

        .lword  dummy_int      ; software int 133
        .lword  dummy_int      ; software int 134
        .lword  dummy_int      ; software int 135
        :
        (省略)
        :
        .lword  dummy_int      ; software int 253
        .lword  dummy_int      ; software int 254
        .lword  dummy_int      ; software int 255
    .endif
    .
    ;-----;
    ; FIXED VECTOR SECTION          ;
    ;-----;
        .section fvector, romdata
        .section fvector, romdata
        .org 0FFFFFFDCH
UDI:      .lword  dummy_int
OVER_FLOW: .lword  dummy_int
BRKI:     .lword  dummy_int
          .lword  0FFFFFFFHH
          .lword  0FFFFFFFHH
WDT:      .lword  dummy_int
          .lword  dummy_int
NMI:      .lword  dummy_int
RESET:    .lword  start

    ;-----;
    ; ID code DEFINITION          ;
    ;-----;
    ; ID code check function
    ; .id "CodeChk"

    ;*****;
    ;                               ;
    ;   End of R32C/100 start up   ;
    ;                               ;
    ;*****;

```

図2.17 スタートアッププログラムsect100.incリスト (12)

2.2.2 スタートアッププログラムのカスタマイズ

(a) スタートアッププログラムの処理概要

(1) ncr0.a30 について

このプログラムは、プログラムの開始時又はリセット直後に実行されます。

このプログラムは主に以下の処理を行います。

- SBDATA領域(SB相対アドレッシングモードでアクセスする領域)の先頭アドレス値(`__SB__`)を設定します。
- プロセッサ動作モードを設定します。
- スタックポインタ(ISPレジスタとUSPレジスタ)の初期化を行います。
- SBレジスタの初期化を行います。
- INTBレジスタの初期化を行います。
- 初期値なしのデータ領域を初期化します。

(1) デフォルト

`bss_NEAR`セクションをゼロクリアします。

(2) `far`修飾した変数を使用する場合

`bss_FAR`セクションをゼロクリアします。

(3) `#pragma SBDATA` 拡張機能を使用する場合

`bss_SB8`セクションをゼロクリアします。

(4) `#pragma SB16DATA` 拡張機能を使用する場合

`bss_SB16`セクションをゼロクリアします。

(5) `#pragma EXTMEM` 拡張機能を使用する場合

`bss_EXT`セクションをゼロクリアします。

(6) `#pragma MONITORn` 拡張機能を使用する場合

`bss_MON1`, `bss_MON2`, `bss_MON3`, `bss_MON4`セクションをゼロクリアします。

- 初期値が格納されたROMセクションから初期値を持つデータ領域へ初期値を転送します。

(1) デフォルト

`data_NEAR_INIT`セクションから`data_NEAR`セクションに初期値を転送します。

(2) `far`修飾された変数を使用する場合

`data_FAR_INIT`セクションから`data_FAR`セクションに初期値を転送します。

(3) `#pragma SBDATA` 拡張機能 を使用する場合

`data_SB8_INIT`セクションから`data_SB8`セクションに初期値を転送します。

(4) `#pragma SB16DATA` 拡張機能 を使用する場合

`data_SB16_INIT`セクションから`data_SB16`セクションに初期値を転送します。

(5) `#pragma EXTMEM` 拡張機能 を使用する場合

`data_EXT_INIT`セクションから`data_EXT`セクションに初期値を転送します。

- (6) #pragma MONITORn 拡張機能 を使用する場合
data_MON1_INIT、data_MON2_INIT、data_MON3_INIT、data_MON4_INITセクションから、それぞれ、data_MON1、data_MON2、data_MON3、data_MON4セクションに初期値を転送します。

- heap領域の初期化を行います。
- 標準入出力関数ライブラリの初期化を行います。
- FBレジスタの初期化を行います。
- main関数の呼び出しを行います。

(b) スタートアッププログラムの変更手順

スタートアッププログラムを、組み込むシステムに合わせて変更する方法の概略を【図 2.18】に示します。

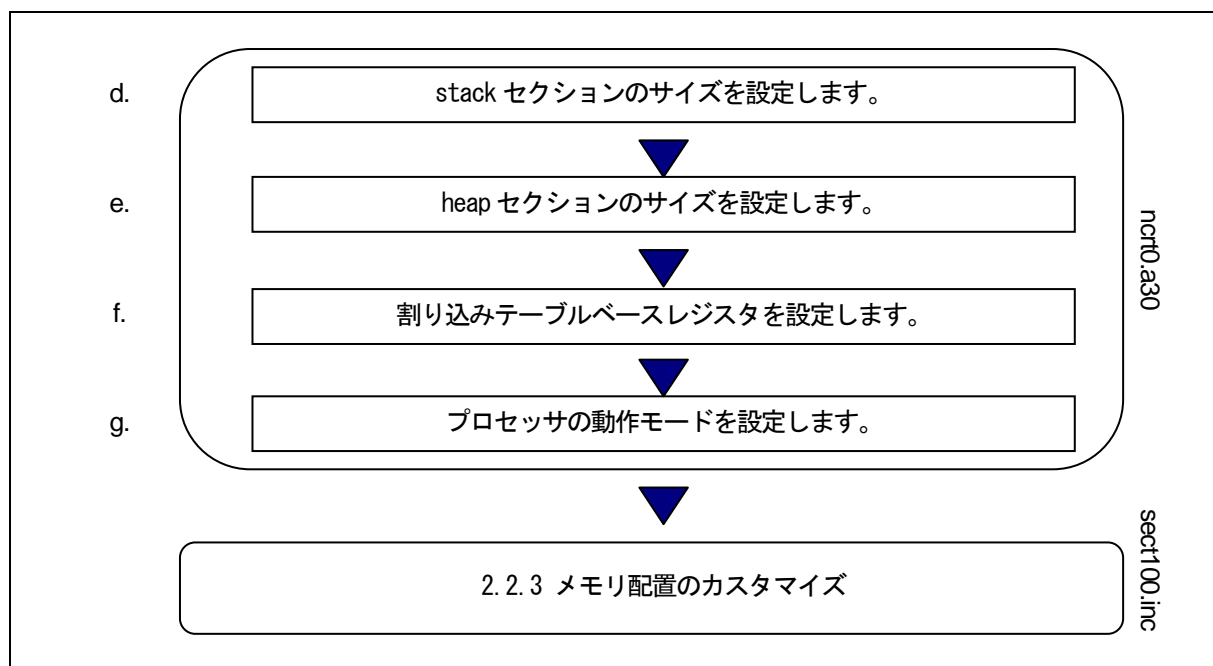


図2.18 スタートアッププログラムの変更手順例

(c) 注意を要するスタートアップの変更例

(1) 標準入出力関数を使用しないときの設定

`_init`関数³は、標準入出力関数ライブラリの入出力の初期化を行います。`_init`関数は、`nct0.a30`内で`main`関数を呼び出す前に呼び出されます。【図 2.19】に`_init`関数の呼び出し部を示します。

アプリケーションプログラム中で標準入出力関数を使用しないときは、`nct0.a30`内の`_init`関数の呼び出し部をコメントにしてください。

³ `_init`関数は、標準入出力関数のためのマイコン（ハードウェア）の初期化も行っています。デフォルトでは、R32C/100を指定した初期化を行っています。標準入出力関数を使用する場合は、組み込むシステムにより`_init`関数等を修正する必要があります。

```

;-----;
; initialize standard I/O      ;
;-----;
.if __STANDARD_IO__ == 1
    .glb          __init
    .call         __init,G
    jsr.a        __init
.endif

```

図2.19 _init関数呼び出し部(ncrt0.a30)

なお、`sprintf`、`scanf` のみを使用する場合、`_init` 関数を呼び出す必要はありません。

(2) メモリ管理関数を使用しないときの設定

メモリ管理関数(`calloc`、`malloc`、等)を使用するために、`heap` セクションの領域の確保に加えて、`ncrt0.a30` 中で以下の設定を行っています。

- (1) 外部変数`char* __mnext`の初期化
heapセクションの先頭アドレスを表すラベル`heap_top`で初期化します。
- (2) 外部変数`unsigned long __msize`の初期化
「2.2.2 e. heapセクションのサイズの設定」で設定した`HEAPSIZE`で初期化します。

【図 2.20】に`ncrt0.a30` 内の初期化部を示します。

```

;-----;
; initialize heap manager      ;
;-----;
.if __HEAP__ != 1
    .glb          __mnext
    .glb          __msize
    mov.l        #heap_top,__mnext
    mov.l        #HEAPSIZE,__msize
.endif

```

図2.20 メモリ管理関数を使用するときの初期化部 (ncrt0.a30)

メモリ管理関数を使用しない場合は、この初期化部をすべてコメントにしてください。コメントにすることで、不要なライブラリがリンクされずROMサイズを節約できます。

(3) 独自の初期化プログラムを記述するときの注意事項

独自の初期化プログラムをスタートアッププログラム中に追加する場合は、以下の点に注意してください。

- (1) 独自の初期化プログラムにおいて、**U**、**B**フラグを変更した場合は、初期化プログラムの出口で**U**、**B**フラグを元の状態に戻してください。また、**SB**レジスタの内容を変更しないでください。
- (2) 独自の初期化プログラムからC言語で記述されたサブルーチン呼び出す場合は、以下の2項に注意してください。
 - **B**、**D**フラグはクリアした状態で呼び出してください。
 - **U**フラグはセットした状態で呼び出してください。

(d) stack セクションのサイズの設定

stack セクションは、ユーザースタック用に使用する領域と割り込みスタック用に使用する領域が含まれます。

スタックは必ず使用しますので、必ず領域を確保してください。スタックサイズは、使用する最大サイズを設定してください。⁴

スタックサイズは、スタックサイズ計算ユーティリティ Call Walker を使用して求めることができます。

(e) heap セクションのサイズの設定

heap セクションのサイズは、プログラム中でメモリ管理関数 `calloc`、`malloc` を使用して確保する最大のメモリ使用量を設定します。メモリ管理関数を使用しないときはサイズを 0 に設定してください。

heap セクションは物理的な RAM 領域を越えないように設定してください。

```
-----  
; HEAP SIZE definition  
-----  
.if __HEAP__ == 1                ; for HEW  
  
HEAPSIZE .equ    0h  
  
.else  
.if __HEAPSIZE__ == 0  
  
HEAPSIZE .equ    300h  
  
.else                ; for HEW  
  
HEAPSIZE .equ    __HEAPSIZE__  
  
.endif  
.endif
```

図2.21 heapサイズの設定例 (ncrt0.a30)

⁴ スタートアッププログラム内でもスタックを使用しています。main()関数を呼び出す前に初期値を再ロードしていますが、main()関数等で使用するスタックサイズが少ない場合は、考慮が必要です。

(f) 割り込みベクタテーブルの設定

nrt0.a30 中の【図 2.22】の部分において割り込みベクタテーブルの先頭アドレスを設定します。設定した値で、INTB レジスタが初期化されます。

```

;-----
; INTERRUPT VECTOR ADDRESS definition
;-----
VECTOR_ADR      .equ      0FFFFFFBDCH

```

図2.22 割り込みベクタテーブルの先頭アドレスの設定例 (nrt0.a30)

サンプルのスタートアッププログラムでは、

0FFFFFFBDCH ~ 0FFFFFFDBH:	割り込みベクタテーブル
0FFFFFFDCH ~ 0FFFFFFFH:	固定ベクタテーブル

で使用するための値を設定しています。通常は、設定値を変更する必要はありません。

(g) プロセッサモードレジスタの設定

nrt0.a30 中の【図 2.23】で示す部分において 04H 番地(プロセッサモードレジスタ)に、組み込むシステムに合わせたプロセッサ動作モードを設定します。

```

;-----;
; after reset, this program will start ;
;-----;
:
(省略)
:
; mov.b      #00H,04H      ; processor mode
:
(省略)
:

```

図2.23 プロセッサモードレジスタ設定例 (nrt0.a30)

プロセッサモードレジスタの詳細については、マイコンのユーザズマニュアルを参照してください。

2.2.3 メモリ配置のカスタマイズ

(a) セクションの構成

ネイティブな環境のコンパイラの場合、コンパイラが生成した実行ファイルはUNIX等のオペレーティングシステムによってメモリ配置が決定されます。しかし、NC100のようなクロス環境のコンパイラでは、ユーザがメモリ配置を決定する必要があります。

NC100は、変数の記憶クラス、初期値を持つ変数、初期値を持たない変数、文字列データ、割り込み処理プログラム、割り込みベクトルアドレステーブル等プログラムの機能ごとにセクションという単位でマイコンのメモリ上に配置します。

セクションを表すセクション名は、セクションベース名とその属性により、【図 2.24】で構成されます。

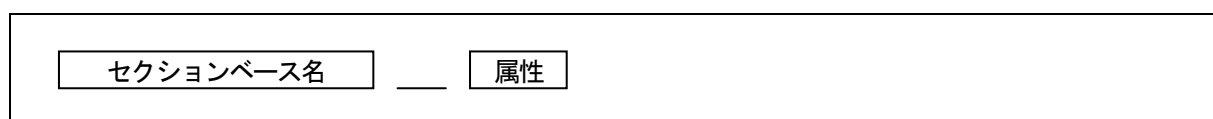


図2.24 セクション名

セクション名を【表 2.10】に属性を【表 2.11】に示します。

表2.10 セクションベース名

セクションベース名	内容
data	初期値のあるデータを格納します。
bss	初期値のないデータを格納します。
rom	文字列、#pragma ROM、const 修飾子で指定されたデータを格納します。

表2.11 属性

属性	意味	対象セクションベース名	
INIT	データの初期値を保持するセクション	data	
NEAR / FAR / SB8 / SB16 / EXT / MON1 / MON2 / MON3 / MON4	NEAR	near属性 ⁵	data, bss, rom
	FAR	far 属性	
	SB8	SBDATA 属性	data, bss
	SB16	SB16DATA 属性	data, bss
	EXT	EXTMEM 属性	data, bss, rom
	MON1	MONITOR1 属性	data, bss
	MON2	MONITOR2 属性	data, bss
	MON3	MONITOR3 属性	data, bss
MON4	MONITOR4 属性	data, bss	

前述の命名規則に準じたセクション以外のセクションの内容を【表 2.12】に示します。

⁵ near, farとは、NC100固有の修飾子です。この修飾子を使用することによりアドレッシングモードを明示的に指定できます。

near.....アクセス範囲は00000000H番地から00007FFFH番地まで、および、0FFFF8000H番地から0FFFFFFFH番地まで

far.....アクセス範囲は00000000H番地から007FFFFFH番地まで、および、0FF80000H番地から0FFFFFFFH番地まで

表2.12 セクションの名称

セクション名	内容
fvector	マイコンの固定ベクタの内容を格納します。
heap	メモリ管理関数(malloc 等)により、プログラム実行中に動的に確保されるメモリ領域です。このセクションはマイコンの任意の RAM 領域に配置できます。
program	program
stack	スタックとして使用する領域です。このセクションはマイコンの任意の RAM 領域に配置できます。
switch_table	switch 文に対するテーブルコードを格納します。 このセクションは、コンパイルオプション"-fswitch_other_section(-fSOS)"を使用した場合のみ生成されます。
vector	マイコンの割り込みベクタテーブルの内容を格納します。割り込みベクタテーブルは intb レジスタ相対によりマイコンの全メモリ空間に任意に配置できます。詳しくはマイコンのハードウェアマニュアルを参照してください。

これらのセクションの配置は、スタートアッププログラムのインクルードファイル `sect100.inc` で設定します。このインクルードファイルの内容を変更することで、セクションの配置を変更することができます。

サンプルのスタートアッププログラムのインクルードファイル `sect100.inc` のセクション配置例を【図 2.25】に示します。

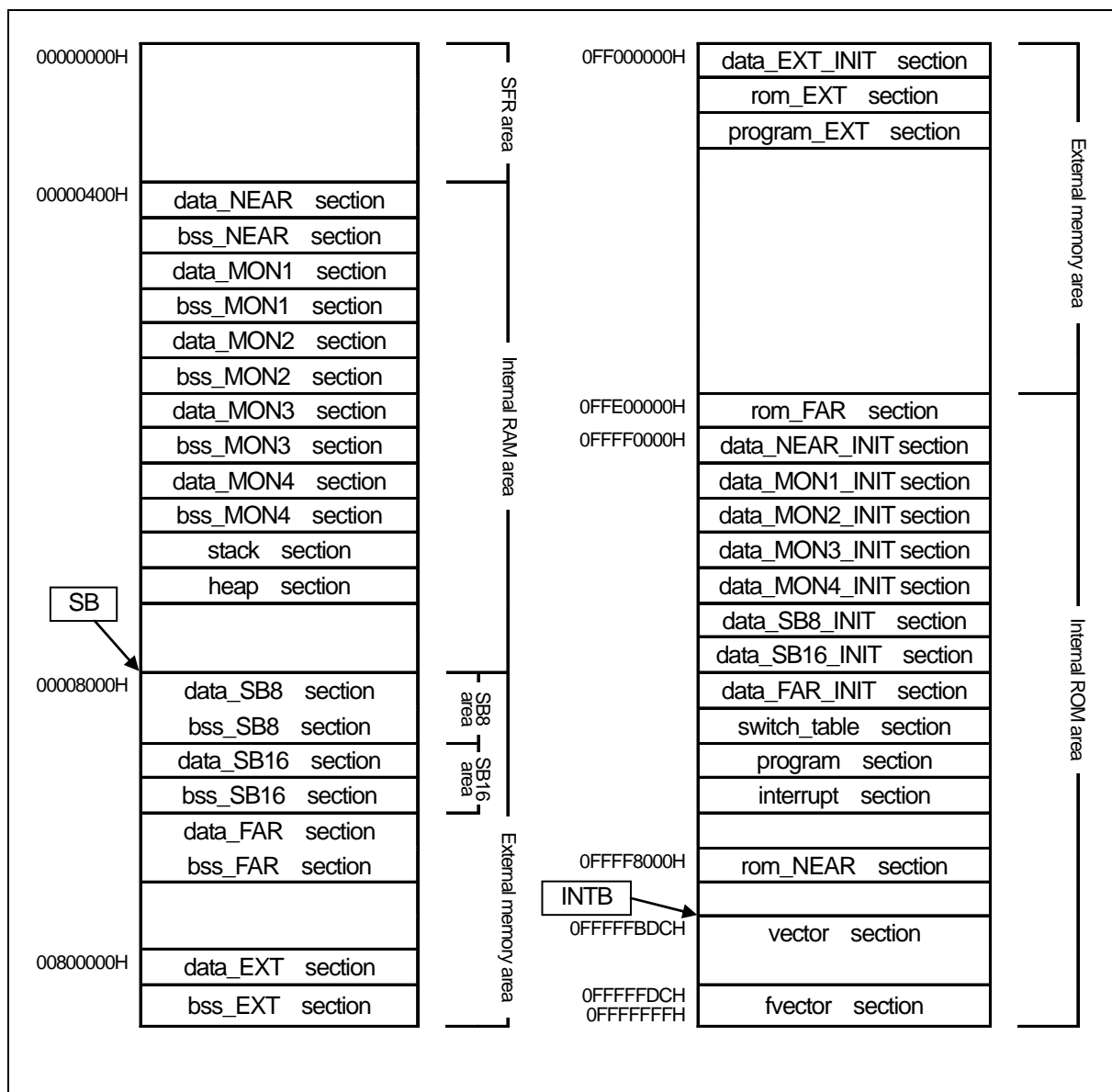


図2.26 セクション配置例

(b) メモリ配置設定用ファイルの概要

(1) sect100.inc について

このプログラムは、nct0.a30 からインクルードされます。このプログラムは主に以下の処理を行います。

- 各セクション配置(順序)の設定を行います。
- セクション開始アドレスの設定を行います。
- stackセクション及びheapセクション領域のサイズを定義します。
- 割り込みベクタテーブルを設定します。
- 固定ベクタテーブルを設定します。

(c) sect100.inc の変更手順

スタートアッププログラムのメモリ配置設定用ファイルを、組み込むシステムに合わせて変更する方法の概略を【図 2.27】に示します。

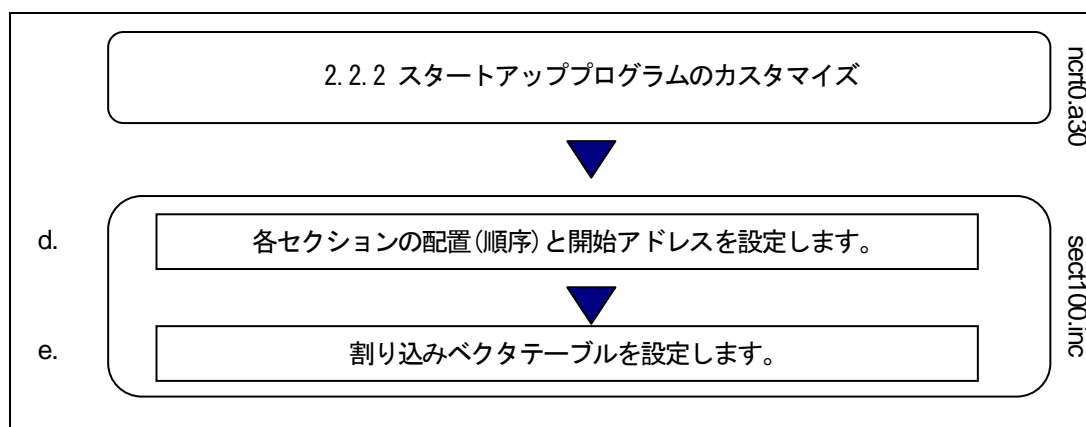


図2.27 メモリ配置の変更手順例

(d) セクション配置(順序)と開始アドレスの設定

セクションの配置(順序)と開始アドレスの設定(プログラム及びデータのROM/RAMへの配置設定)は、スタートアッププログラムのインクルードファイル sect100.inc で行います。

セクションの配置は、sect100.incに定義された順に配置されます。また、そのセクションの開始アドレスはas100の指示命令".ORG"を用いて指定します。【図 2.28】に設定例を示します。

```

.section      program, code, align
.org         0FFFF0000H      ← program セクションの開始アドレスの設定
  
```

図2.28 セクション開始アドレスの設定例

セクションに対する開始アドレスの指定がない場合、前に定義したセクションに連続してメモリに配置されます。

(1) セクションの配置規則

セクションには、マイコンのメモリ属性(RAM/ROM)に影響されるため、配置できる領域が限られているセクションがあります。セクションの配置は、以下の規則に従ってください。

(1) RAM領域に配置するセクション

- | | |
|------------------|-----------------|
| ● stackセクション | ● heapセクション |
| ● data_NEARセクション | ● bss_NEARセクション |
| ● data_FARセクション | ● bss_FARセクション |
| ● data_EXTセクション | ● bss_EXTセクション |
| ● data_SB8セクション | ● bss_SB8セクション |
| ● data_SB16セクション | ● bss_SB16セクション |
| ● data_MON1セクション | ● bss_MON1セクション |
| ● data_MON2セクション | ● bss_MON2セクション |
| ● data_MON3セクション | ● bss_MON3セクション |
| ● data_MON4セクション | ● bss_MON4セクション |

(2) ROMに配置するセクション

- | | |
|-----------------------|-----------------------|
| ● programセクション | ● program_EXTセクション |
| ● interruptセクション | ● fvectorセクション |
| ● switch_tableセクション | ● rom_FARセクション |
| ● rom_NEARセクション | ● rom_EXTセクション |
| ● data_NEAR_INITセクション | ● data_MON1_INITセクション |
| ● data_FAR_INITセクション | ● data_MON2_INITセクション |
| ● data_EXT_INITセクション | ● data_MON3_INITセクション |
| ● data_SB8_INITセクション | ● data_MON4_INITセクション |
| ● data_SB16_INITセクション | |

また、セクションはマイコンのメモリ空間に配置できる領域が限られているセクションがあります。

(1) 0H~07FFFH、0FFFF8000H~0FFFFFFFH(near領域)にのみ配置できるセクション

- | | |
|------------------|-----------------|
| ● data_NEARセクション | ● bss_NEARセクション |
| ● rom_NEARセクション | |

(2) 0H~07FFFFH、0FF80000H~0FFFFFFFH(far領域)にのみ配置できるセクション

- | | |
|---------------------|------------------|
| ● programセクション | ● interruptセクション |
| ● switch_tableセクション | ● bss_FARセクション |
| ● data_FARセクション | ● bss_MON1セクション |
| ● data_MON1セクション | ● bss_MON2セクション |
| ● data_MON2セクション | ● bss_MON3セクション |
| ● data_MON3セクション | ● bss_MON4セクション |
| ● data_MON4セクション | ● vectorセクション |
| ● rom_FARセクション | |

- (3) 0FFFFFFDCH~0FFFFFFFHにのみ配置できるセクション
- fvectorセクション
- (4) R32C/100シリーズの全メモリ空間に配置できるセクション
- stackセクション
 - data_EXTセクション
 - rom_EXTセクション
 - data_NEAR_INITセクション
 - data_EXT_INITセクション
 - data_SB8_INITセクション
 - data_MON1_INITセクション
 - data_MON3_INITセクション
 - heapセクション
 - bss_EXTセクション
 - program_EXTセクション
 - data_FAR_INITセクション
 - data_SB16_INITセクション
 - data_MON2_INITセクション
 - data_MON4_INITセクション

また、以下のデータに関するセクションのサイズが0の場合、必ずしも定義する必要はありません。

- program_EXTセクション
- data_NEARセクション
- data_FARセクション
- data_EXTセクション
- data_MON1セクション
- data_MON2セクション
- data_MON3セクション
- data_MON4セクション
- data_SB8セクション
- data_SB16セクション
- bss_NEARセクション
- bss_FARセクション
- bss_EXTセクション
- bss_SB8セクション
- bss_SB16セクション
- rom_NEARセクション
- switch_tableセクション
- data_NEAR_INITセクション
- data_FAR_INITセクション
- data_EXT_INITセクション
- data_MON1_INITセクション
- data_MON2_INITセクション
- data_MON3_INITセクション
- data_MON4_INITセクション
- data_SB8_INITセクション
- data_SB16_INITセクション
- bss_MON1セクション
- bss_MON2セクション
- bss_MON3セクション
- bss_MON4セクション
- rom_EXTセクション
- rom_FARセクション

(2) シングルチップモードにおけるセクション配置例

シングルチップモードにおけるセクション配置を行うインクルードファイルsect100.incの記述例を【図 2.29】、～【図 2.32】に示します。

```

;*****;
;
; C COMPILER for R32C/100
; Copyright(C) XXXX. Renesas Electronics Corp.
; and Renesas Solutions Corp., All rights reserved.
;
; (省略)
;
; $Id: sect100.inc,v X.X XXXX/XX/XX XX:XX:XX XXXXX Exp $
;
;*****;
;-----;
;
; Arrangement of section
;
;-----;
;-----;
; NEAR RAM SECTIONS ;
;-----;
;
; .section data_SB8, data
; .org 00000400H
data_SB8_top:
; .section bss_SB8, data, align
bss_SB8_top:
; .section data_NEAR, data, align
data_NEAR_top:
; .section bss_NEAR, data, align
bss_NEAR_top:
; .section data_MON1, data, align
data_MON1_top:
; .section bss_MON1, data, align
bss_MON1_top:
; .section data_MON2, data, align
data_MON2_top:
; .section bss_MON2, data, align
bss_MON2_top:
; .section data_MON3, data, align
data_MON3_top:
; .section bss_MON3, data, align
bss_MON3_top:
; .section data_MON4, data, align
data_MON4_top:
; .section bss_MON4, data, align
bss_MON4_top:
;-----;
; STACK SECTION ;
;-----;
; .section stack, data, align
; .blkb STACKSIZE
; .align
stack_top:
; .blkb ISTACKSIZE
; .align
istack_top:

```

図2.29 シングルチップモードにおけるsect100.incリスト(1)

```

;-----;
; HEAP SECTION ;
;-----;

heap_top:      .section heap,      data, align
               .blkb             HEAPSIZE

;-----;
; SB RELATIVE RAM SECTIONS ;
;-----;
;               .section data_SB8,  data
;               .org                00008000H
;data_SB8_top:
;               .section bss_SB8,   data, align
;bss_SB8_top:
;               .section data_SB16, data, align
;data_SB16_top:
;               .section bss_SB16,  data, align
;bss_SB16_top:

;-----;
; FAR RAM SECTIONS ;
;-----;
;               .section data_FAR,  data, align
data_FAR_top:  .section bss_FAR,   data, align
bss_FAR_top:

;-----;
; EXTENDED RAM SECTIONS ;
;-----;
;               .section data_EXT,  data
;               .org                00800000H
data_EXT_top: .section bss_EXT,   data, align
bss_EXT_top:

;-----;
; EXTENDED ROM SECTIONS ;
;-----;
;               .section data_EXT_INIT, romdata
;               .org                0FF000000H
data_EXT_INIT_top:
               .section rom_EXT,  romdata, align
rom_EXT_top:
               .section program_EXT, code, align

;-----;
; FAR ROM SECTIONS ;
;-----;
;               .section rom_FAR,  romdata
;               .org                0FFE00000H
rom_FAR_top:

```

← セクションサイズがゼロですので、取り除く事ができます。

この場合、nrcrt0.a30 の該当領域の初期化プログラムも取り除く必要があります。

図2.30 シングルチップモードにおけるsect100.incリスト(2)

```

;-----;
; INITIAL DATA SECTIONS
;-----;
        .section data_NEAR_INIT, romdata
        .org    0FFFF0000H
data_NEAR_INIT_top:
        .section data_MON1_INIT, romdata, align
data_MON1_INIT_top:
        .section data_MON2_INIT, romdata, align
data_MON2_INIT_top:
        .section data_MON3_INIT, romdata, align
data_MON3_INIT_top:
        .section data_MON4_INIT, romdata, align
data_MON4_INIT_top:
        .section data_SB8_INIT, romdata, align
data_SB8_INIT_top:
;
; .section data_SB16_INIT, romdata, align
;data_SB16_INIT_top:
        .section data_FAR_INIT, romdata, align
data_FAR_INIT_top:

;-----;
; SWITCH TABLE SECTIONS
;-----;
        .section switch_table, romdata, align

;-----;
; CODE SECTIONS
;-----;
        .section program, code, align

        .section interrupt, code, align

;-----;
; NEAR ROM SECTIONS
;-----;
;
; .section rom_NEAR, romdata
; .org    0FFFF8000H
;rom_NEAR_top:

;-----;
; VARIABLE VECTOR SECTION
;-----;
        .section vector, romdata
        .org    VECTOR_ADR
.if    __MVT__ == 1
:
(省略)
:
.endif

```

図2.31 シングルチップモードにおけるsect100.incリスト(3)


```

;-----;
; FIXED VECTOR SECTION ;
;-----;
                .section   fvector,      romdata
                .org       0FFFFFFDCH
UDI:            .lword     dummy_int
OVER_FLOW:     .lword     dummy_int
BRKI:          .lword     dummy_int
                .lword     0FFFFFFFHH
                .lword     0FFFFFFFHH
WDT:           .lword     dummy_int
                .lword     dummy_int
NMI:           .lword     dummy_int
RESET:         .lword     start

;-----;
; ID code DEFINITION ;
;-----;
; ID code check function
; .id "CodeChk"

;*****;
; ;
;   End of R32C/100 start up ;
; ;
;*****;

```

図2.32 シングルチップモードにおけるsect100.incリスト(4)

(e) 割り込みベクタテーブルの設定

割り込み処理を使用するプログラムでは、

- (1) sect100.inc中のvectorセクションの割り込みベクタテーブルの設定を行なうことで、割り込みベクタテーブルの設定してください。

割り込みベクタの内容は、マイコンの機種により異なります。使用するマイコン機種に合わせて設定する必要があります。詳細については、各機種のユーザーズマニュアルを参照してください。

(1) sect100.inc で割り込みベクタテーブルの設定

割り込み処理を使用するプログラムでは、sect100.inc 中の vector セクションの割り込みベクタテーブルを変更します。

【図 2.33】に割り込みベクタテーブル例を示します。

```

;-----;
; VARIABLE VECTOR SECTION ;
;-----;
                .section vector,          romdata
                .org      VECTOR_ADR
.if      _MVT_ == 1
    .lword  dummy_int      ; BRK                (software int 0)
    :
    (省略)
    :
    .lword  dummy_int      ; DMA0                (software int 8)
    .lword  dummy_int      ; DMA1                (software int 9)
    .lword  dummy_in      ; DMA2                (software int 10)
    :
    (省略)
    :
    .lword  dummy_int      ; uart1 trance/NACK    (software int 19)
    .lword  dummy_int      ; uart1 receive/ACK    (software int 20)
    .lword  dummy_int      ; TIMER B0            (software int 21)
    :
    (省略)
    :
    .lword  dummy_int      ; INT5                (software int 26)
    .lword  dummy_int      ; INT4                (software int 27)
    :
    (省略)
    :
    .lword  dummy_int      ; uart2 trance/NACK    (software int 33)
    .lword  dummy_int      ; uart2 receive/ACK    (software int 34)
    :
    (省略)
    :
    .lword  dummy_int      ; software int 255

※dummy_int はダミーの割り込み処理関数です

```

図2.33 割り込みベクタテーブルの設定例

次の手順で sect100.inc 中の vector セクションの割り込みベクタテーブルを変更します。

- (1) 割り込み処理関数名をアセンブラの指示命令.GLBで外部参照宣言します。
- (2) 本コンパイラで作成した割り込み処理関数への登録ラベル名は、関数名の前に_(アンダースコア)が付加されます。したがって、ここで宣言する割り込み処理関数名にもアンダースコアを付加して記述します。
- (3) 使用する割り込みも該当する割り込みベクタテーブルのダミー割り込み関数名dummy_intから使用する割り込み処理関数名に置き換えます。

【図 2.34】にUART1 送信割り込み処理関数uarttmの設定例を示します。

```

    .lword  dummy_int      ; uart0 receive (for user)
    .glb   _uarttrn
    .lword  _uarttrn      ; uart1 trance (for user)
    (以下省略)

```

← 上記(1)の処理
← 上記(2)の処理

図2.34 割り込みベクタテーブルの設定例

第3章 プログラミング

この章では、本コンパイラを使用してプログラミングを行う上で、注意すべき事項等について説明します。

3.1 注意事項

本資料に記載の製品データ、図、表に示す技術的な内容、プログラムおよ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。株式会社ルネサス エレクトロニクスおよび株式会社ルネサス ソリューションズは、適用可否に対する責任は負いません。

3.1.1 コンパイラのバージョンアップ等についての注意事項

本コンパイラが生成する機械語命令(アセンブリ言語)は、コンパイル時に指定する起動オプション、バージョンアップの内容等により変化します。したがって、起動オプションの変更又はコンパイラのバージョンアップを行った場合は再度アプリケーションプログラムの動作評価を必ず行ってください。

また、割り込み処理プログラムと被割り込み処理プログラム間、リアルタイム OS 上のタスク間等で、同じ RAM データを参照し内容を変更する場合は、必ず `volatile` 指定等の排他制御を行ってください。また、ビットフィールド構造体において、メンバ名が異なっている場合においても、同一の RAM 上に確保される場合は、同様に排他制御を行ってください。

3.1.2 マイコンの機種依存部に関する注意事項

SFR 領域のレジスタへの書き込み、または読み出しには特定の命令を使用しなければならないことがあります。この特定の命令は機種ごとに異なりますので、詳しくは各マイコン機種のユーザーズマニュアルを参照してください。

本コンパイラは、SFR 領域のレジスタへの書き込み、読み出しには、使用できない命令を生成する場合があります。

C 言語で SFR 領域のレジスタへの書き込み、読み出しをする場合は、`asm` 関数を使用してプログラム中に直接命令を記述してください。この場合、コンパイラのバージョン、オプションの有無に関わらず、生成されたコードに問題が無いことを必ず確認してください。

【図 3.1】のような C 言語記述を SFR 領域に行った場合、割り込み要求ビットの判定が正常に行われず意図しない動作を行う可能性があります。

```

#pragma ADDRESS TA0IC 006Ch      /*R32C/100 タイマ A0 割込み制御レジスタ */

struct {
    char    ILVL : 3;
    char    IR : 1;          /* 割込み要求ビット */
    char    dmy : 4;
} TA0IC;

void    wait_until_IR_is_ON(void)
{
    while(TA0IC.IR == 0)      /* 1 になるまで待つ */
    {
        ;
    }
    TA0IC.IR = 0;            /* 1 になったら 0 に戻す */
}

```

図3.1 SFR領域に対するCソースコード記述

3.1.3 最適化について

(a) 常に行われる最適化

最適化オプションの有無に関わらず、以下のものは最適化されます。

(1) 意味のない変数アクセス

例えば、以下の【図 3.2】に示される変数`port`は、読み出し結果を使用しないため、読み出し動作が削除されます。

```

extern int    port;

void    func(void)
{
    port;
}

```

図3.2 意味のない変数アクセス例 (最適化される)

この例は`port`を読み出すだけの操作を行いたいことを意図して記述したのですが、実際には読み出すコードは最適化されて出力されません。最適化を行わないようにするには【図 3.3】に示すように`volatile`修飾子を付加してください。

```
extern int volatile      port;

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

図3.3 意味のない変数アクセス例 (最適化を抑制)

(2) 意味のない比較

```
int    func(char c)
{
    int    i;

    if(c != -1)
        i = 1;
    else
        i = 0;
    return i;
}
```

図3.4 意味のない比較

この例の場合、変数 `c` は `char` と記述されていますので、本コンパイラのデフォルトでは `unsigned char` 型として取り扱います。`unsigned char` 型で表現できる数値の範囲は 0 から 255 までですので、変数 `c` は -1 の値を持つことはありません。このため、このような論理的に意味のない文を記述された場合、本コンパイラではアセンブラコードを生成しません。

(3) 実行されることのないプログラム

論理的に実行されることのないプログラムに対するアセンブラコードは、生成されません。

```
void    func(int i)
{
    func2(i);
    return;

    i = 10;          ← 実行されることがない部分
}
```

図3.5 実行されることのないプログラム

(4) 定数間の演算

定数間の演算はコンパイル時に演算されます。

```
int    func(void)
{
    int    i = 1 + 2;           ← コンパイル時に演算されます
    return i;
}
```

図3.6 定数間の演算

(5) 最適命令の選択

乗除算をシフト命令で出力する等の最適命令の選択は、最適化オプションの有無に関わらず、常に行われます。

(b) volatile 修飾子について

volatile 修飾子を使用することにより、変数の参照や参照順序、参照回数等に対して最適化の影響が無いようにすることができます。ただし、以下の図に示すような、解釈が曖昧になるような記述は行わないでください。

```
int    a;
int volatile    b, c;

a = b = c;           /* a = c なのか、a = b なのか? */
a = ++b;            /* a = b なのか、a = (b+1) なのか? */
```

図3.7 volatile修飾子の解釈が曖昧になる例

3.1.4 register変数の使用に関する注意事項

(a) register 修飾とコンパイルオプション"-fenable_register(-fER)"について

コンパイルオプション"-fenable_register (-fER)"を指定することにより、特定条件を満たす register 修飾を行った変数を強制的にレジスタに割り付けることができます。

この機能は、最適化に頼らずに生成コードを改良するためのものです。むやみに使用すると逆効果となりますので、必ず生成コードを確認した上で、使用してください。

(b) register 修飾と自動的なレジスタ割り付けについて

本コンパイラでは、自動的に変数のレジスタへの割り付けを行います。この割り付けの機能には、register 修飾を行っているか否かは影響しません。

3.1.5 スタートアップの扱いについて

ご使用のマイコン機種、お客様のシステムによりスタートアップを変更していただく必要があります。

機種により変更を必要とする内容は対応機種のデータブック等を参照いただき、添付のスタートアップファイルを修正して、ご使用ください。

3.2 生成コードの向上のために

3.2.1 コード効率の良いプログラミング方法

(a) プロトタイプ宣言の活用

本コンパイラでは、関数のプロトタイプ宣言を行なうことにより、効率の良い関数呼び出しを行なうことができます。すなわち、本コンパイラでは関数のプロトタイプ宣言を行なわない場合、その関数を呼び出すときに、その関数の引数を【表 3.1】に示す規則によりスタック領域に積みます。

表3.1 引き数に関するスタックの使用規則

データ型	スタックに積むときの規則
char 型 short 型	int 型に拡張して積む。
float 型	double 型に拡張して積む。
その他の型	型の拡張は行わずに積む。

このため、関数のプロトタイプ宣言を行なわない場合は、冗長な型拡張を行なう場合があります。

関数のプロトタイプ宣言を行なうことにより、これらの冗長な型拡張を抑止し、また、レジスタに引数を割り当てることが可能になるため、効率の良い関数呼び出しを行なうことができます。

(b) SB レジスタの活用

SBレジスタ¹を用いたアドレッシングモードを使用することにより、アプリケーションプログラムサイズ(ROMサイズ)を削減することができます。本コンパイラでは【図 3.8】で示す記述を行なうことにより、SBレジスタを用いたアドレッシングモードを使用する変数を宣言することができます。

```
#pragma SBDATA val
int    val;
```

図3.8 SBレジスタを用いたアドレッシングモードを使用する変数の宣言例

(c) コンパイルオプション"-fJSRW"による ROM サイズ圧縮

本コンパイラでは、ファイル外で定義された関数を呼び出す場合は、"JSR.A"命令で呼び出しを行ないます。しかし、プログラムサイズがあまり大きくない場合、大半の関数が"JSR.W"命令で呼び出せる場合があります。

このような場合にコンパイルオプション"-fJSRW"を指定してコンパイルをおこない、リンク時にエラーの発生した関数のみを"#pragma JSRA 関数名"を用いて宣言することにより ROM サイズの圧縮が期待できます。

※コンパイルオプション"-OGJ"を使用すると、リンク時に最適な jmp 命令を選択します。

¹ 本コンパイラでは、SBレジスタはリセット後に初期化を行ない、以降は固定で使用することを前提としています。

(d) その他

その他の方法として次のような記述の変更を行うことにより、ROM 容量を圧縮できる場合があります。

- (1) 一回しか呼ばれない比較的小さい関数をinline関数にする。
- (2) if-else文をswitch文で置き換える(判定対象の変数が配列、ポインタ、構造体などの単純な変数ではない場合に効果があります)。
- (3) char型の範囲でしか値を返さない関数の、戻り値の型を char型で宣言する。
- (4) 関数呼び出しをまたいで使用する変数をレジスタ変数にしない。

3.2.2 スタートアップ処理を高速化する方法

スタートアッププログラム ncr10.a30 には bss 領域のクリア処理が含まれています。この処理は C 言語の言語仕様として初期化されていない変数は初期値として 0 を持つという規格を満たすための処理です。

例えば、【図 3.9】に示す記述の場合、初期値を記述していませんので、スタートアップ処理時に初期値として 0 を与える処理(bss領域²のクリア処理)が必要になります。

```
static int      i;
```

図3.9 初期値を持たない変数の宣言例

応用によっては初期値を持たない変数を 0 クリアする必要が無いものがあります。この場合はスタートアッププログラム内の bss 領域のクリア処理部をコメントアウトすれば、スタートアップ処理を高速化することができます。

```

;-----;
; zero clear BSS ;
;-----;
; BZERO      bss_SB8_top, bss_SB8
; BZERO      bss_SB16_top, bss_SB16
; BZERO      bss_NEAR_top, bss_NEAR
; BZERO      bss_FAR_top, bss_FAR
; BZERO      bss_MON1_top, bss_MON1
; BZERO      bss_MON2_top, bss_MON2
; BZERO      bss_MON3_top, bss_MON3
; BZERO      bss_MON4_top, bss_MON4

```

図3.10 bss領域のクリア処理のコメントアウト例

² 初期値を持たない RAM上の外部変数のことを"bss"と呼びます。

3.3 アセンブリ言語プログラムとの結合方法

3.3.1 C言語プログラムからアセンブラ関数の呼び出し方法

(a) 引数のないアセンブラ関数の呼び出し方法

C 言語プログラムからアセンブラ関数を呼び出す場合は、C 言語で記述した関数呼び出しと同様にアセンブラ関数名で呼び出します。

アセンブラ関数の先頭ラベル名は名前の最初に_(アンダースコア)を付加する必要があります。C 言語プログラムからアセンブラ関数を呼び出すときは、アセンブラ関数の名前(先頭ラベル名)から、アンダースコアを除いた名前を使用します。呼び出す C 言語プログラム中には、必ずアセンブラ関数のプロトタイプ宣言を記述してください。

【図 3.11】 にアセンブラ関数 `asm_func` を呼び出すときの記述例を示します。

```
extern void    asm_func( void );           ← アセンブラ関数のプロトタイプ宣言

void          main()
{
    :
    (省略)
    :
    asm_func();                            ← アセンブラ関数の呼び出し
}
```

図3.11 引数がない場合のアセンブラ関数の呼び出し例 (smp1.c)

```
_main:      .glb    _main
            :
            (省略)
            :
            jsr    _asm_func              ← アセンブラ関数の呼び出し('_'を付加しています)
            rts
```

図3.12 図3.12 smp1.cのコンパイル結果(抜粋) (smp1.a30)

(b) アセンブラ関数に対して引数を与える場合

アセンブラ関数に引数を渡す場合、拡張機能の`#pragma PARAMETER`を使用します。

`#pragma PARAMETER`は、32bit 汎用レジスタ(R2R0、R3R1、R6R4、R7R5)、16bit 汎用レジスタ(R0、R1、R2、R3、R4、R5、R6、R7)、8bit 汎用レジスタ(R0L、R0H、R1L、R1H、R2L、R2H、R3L、R3H)、及び、アドレスレジスタ(A0、A1、A2、A3)を介して、アセンブラ関数に引数を渡します。

64bitの場合、32bit 汎用レジスタの対(R3R1R2R0、R7R5R6R4)、または、アドレスレジスタの対(A1A0、A3A2)を使用します。

`#pragma PARAMETER`でアセンブラ関数を呼び出す手順を以下に示します。

- (1) `#pragma PARAMETER`宣言を記述する前にアセンブラ関数のプロトタイプ宣言を行います。このときには、必ず引数の型宣言を行なってください。
- (2) `#pragma PARAMETER`でアセンブラ関数の引数リストに使用するレジスタ名を宣言します。

【図 3.13】に`#pragma PARAMETER`を使用したアセンブラ関数`asm_func`を呼び出すときの記述例を示します。

```
extern short asm_func(short, short);
#pragma PARAMETER asm_func(R0, R1)

void main(void)
{
    short i = 0x02;
    short j = 0x05;

    asm_func(i, j);
}
```

← 引数を R0、R1 レジスタを介してアセンブラ関数に渡しています

図3.13 引数がある場合のアセンブラ関数の呼び出し例 (smp2.c)

```

        .SECTION program, CODE, ALIGN
        ._file 'smp2.c'
        .align
        ._line 5
;## # C_SRC : {
        .glob _main
_main:
        ._line 6
;## # C_SRC :          short i = 0x02;
        mov.w #0002H,R0      ; i      ← 引数を R0、R1 レジスタを介してアセンブラ関数に
        ._line 7                                渡しています
;## # C_SRC :          short j = 0x05;
        mov.w #0005H,R1      ; j
        ._line 9
;## # C_SRC :          asm_func(i, j);
        jsr _asm_func        ← アセンブラ関数の呼び出し('_'を付加しています)
        ._line 10
;## # C_SRC :      }
        rts

```

図3.14 smp2.cのコンパイル結果(抜粋)(smp2.a30)

(c) #pragma PARAMETER 宣言における引数型及び戻り値型の制限

#pragma PARAMETER 宣言で構造体型、共用体型の引数は宣言できません。
また、アセンブラ関数の戻り値として構造体型、共用体型の戻り値は定義できません。

3.3.2 アセンブラ関数の記述方法

(a) 呼び出されるアセンブラ関数の記述方法

アセンブラ関数の入り口処理の記述手順を以下に示します。

- (1) アセンブラの指示命令".SECTION"でセクション名を指定します。
- (2) 関数名ラベルをアセンブラの指示命令".GLB"でグローバル指定します。
- (3) 関数名に_(アンダースコア)を付加して、ラベルとして記述します。
- (4) 関数内でB及びUフラグを変更する場合は、フラグレジスタをスタック上に退避してください。³

アセンブラ関数の出口処理の記述手順を以下に示します。

- (5) 関数内でB及びUフラグを変更した場合は、スタックからフラグレジスタを復帰してください。
- (6) RTS命令を記述します。

また、アセンブラ関数内でSB、FBレジスタ内容を書き換える操作は行わないでください。

SB、FBレジスタの内容を書き換える場合は、関数の入口でスタックに退避し、関数の出口でスタックから復帰してください。

【図3.15】にアセンブラ関数の記述例を示します。この例では、セクション名を本コンパイラが出力するセクション名と同じprogramを用いています。

```

        .section      program, code, align    ← (1)
        .glb         _asm_func              ← (2)
_asm_func:                                         ← (3)
        pushc       FLG                    ← (4)
        mov.l      SYM1, R3R1

        popc       FLG                    ← (5)
        rts         ← (6)
        .END

```

図3.15 アセンブラ関数の記述例

³ アセンブラ関数内では、B及びUフラグの変更は行なわないでください。

(b) アセンブラ関数からの戻り値の返し方

アセンブラ関数からC言語プログラムに値を返す場合、整数型、ポインタ型、浮動小数点型については、レジスタ渡しで戻り値を返すことができます。【表 3.2】に戻り値に関する呼び出し規則を、【図 3.16】に戻り値を返すアセンブラ関数の記述例を示します。

表3.2 戻り値に関する呼び出し規則

戻り値の型	規則
char 型 _Bool 型	R0L レジスタ
int 型 (16 ビット) short int 型	R0 レジスタ
int 型 (32 ビット) float 型 long 型	R2R0 レジスタ
ポインタ型	A0 レジスタ
long long 型 double 型 long double 型	A1A0 レジスタ (上位 32 ビットは A1 レジスタに、下位 32 ビットは A0 レジスタに格納)
構造体型 共用体型	呼び出しを行う直前に、戻り値を格納するための領域を指す far アドレスをスタックに積みま す。呼び出された関数はリターンする前にスタックに積まれた far アドレスで指す領域に戻り値 を書き込みます。

```

        .section program
        .glob      _asm_func
_asm_func:
        :
        (省略)
        :
        mov.l     #01A000H, R2R0
        rts
        .END

```

図3.16 long型の戻り値を返すアセンブラ関数の記述例

(c) C 言語の変数の参照方法

アセンブラ関数はC言語プログラムとは別のファイルに記述するため、C言語の大域変数のみ参照することができます。

C言語の変数名をアセンブラ関数内で記述するときは、変数名の前に_(アンダースコア)を付加します。また、アセンブリ言語プログラムでは外部参照する変数をアセンブラの指示命令.GLBで外部参照宣言する必要があります。

【図 3.17】にC言語プログラムの大域変数counter をアセンブラ関数asm_func内で参照する例を示します。

```

C 言語プログラム:

unsigned short  counter;          ← C 言語プログラムの大域変数

void  main(void)
{
    :
    (省略)
    :
}

アセンブラ関数:

        .glb      _counter          ← C 言語プログラムの大域変数を外部参照宣言
_asm_func:
    :
    (省略)
    :
    mov.w      _counter, R0        ← 参照

```

図3.17 C言語の大域変数の参照方法

(d) 割り込み処理をアセンブラ関数で記述するときの注意事項

割り込み処理を実行するプログラム(関数)では、出入り口で以下の処理を行う必要があります。

- (1) 関数の入口でレジスタ(R2R0、R3R1、R6R4、R7R5、A0、A1、A2、A3)を一括に退避します。
- (2) 関数の出口でレジスタ(R2R0、R3R1、R6R4、R7R5、A0、A1、A2、A3)を一括に復帰します。
- (3) 関数からのリターンにREIT命令を使用します。

【図3.18】に割り込み処理のアセンブラ関数の記述例を示します。

```

        .section program
        .glb      _func
_int_func:
    pushm      R2R0,R3R1,R6R4,R7R5,A0,A1,A2,A3  ← レジスタの一括退避
    mov.b      #01H, R0L
    :
    (省略)
    :
    popm       R2R0,R3R1,R6R4,R7R5,A0,A1,A2,A3  ← レジスタの一括復帰
    reit                          ← C 言語プログラムへリターン
    .END

```

図3.18 割り込み処理のアセンブラ関数の記述例

(e) アセンブラから C 言語関数を呼び出すときの注意事項

アセンブリ言語プログラムから C 言語で記述された関数を呼び出す場合は、以下の点に注意してください。

- (1) C 言語の関数名に_(アンダースコア)あるいは\$(ダラー)を付加したラベル名で呼び出してください。
- (2) アセンブラ関数中で使用しているレジスタは C 言語関数を呼び出す前に退避し、C 言語関数から戻った後に復帰してください。

3.3.3 アセンブラ関数の記述に関する注意事項

C 言語プログラムから呼び出すアセンブリ言語の関数(サブルーチン)を記述する場合、以下の点に注意してください。

(a) B、U フラグの取り扱いに関する注意事項

アセンブラ関数から C 言語プログラムにリターンするときは、必ず B フラグ及び U フラグを呼び出し時と同じ状態にしてください。

(b) FB レジスタの取り扱いに関する注意事項

アセンブラ関数の中で FB(フレームベースレジスタ)の値を変更した場合、呼び出し元の C 言語プログラムへ正常に復帰できなくなります。したがって、アセンブラ関数中で FB の値を変更しないでください。システム的设计上やむをえず変更する場合は、関数の先頭でスタックに退避して、リターンするときに復帰させてください。

(c) 汎用レジスタ及びアドレスレジスタの取り扱いに関する注意事項

アセンブラ関数の中で汎用レジスタ(R2R0、R3R1、R6R4、R7R5)及びアドレスレジスタ(A0、A1、A2、A3)の内容を変更しても問題はありません。

(d) アセンブラ関数への引数に関する注意事項

アセンブリ言語で記述した関数に対して引数を渡す場合、`#pragma PARAMETER`機能を使用してその引数をレジスタを介して渡すことができます。その書式を【図 3.19】に示す(図中の `asm_func` はアセンブラ関数名です)。

```
short    asm_func(short, short);           ← アセンブラ関数のプロトタイプ宣言
#pragma PARAMETER asm_func(R0, R1)
```

図3.19 アセンブラ関数の記述例

`#pragma PARAMETER` は、32 ビット汎用レジスタ(R2R0、R3R1、R6R4、R7R5)、16 ビット汎用レジスタ(R0、R1、R2、R3、R4、R5、R6、R7)、8 ビット汎用レジスタ(R0L、R0H、R1L、R1H、R2L、R2H、R3L、R3H)及びアドレスレジスタ(A0、A1、A2、A3)を介してアセンブラ関数に引数を渡します。また、32 ビット汎用レジスタ及びアドレスレジスタを組み合わせるとしてアセンブラ関数に引数を渡します。なお、`#pragma PARAMETER` 宣言の前には必ずアセンブラ関数のプロトタイプ宣言を行なってください。

ただし、`#pragma PARAMETER` 宣言で構造体型、共用体型は宣言することはできません。

また、アセンブラ関数の戻り値として構造体型、共用体型の戻り値は定義できません。

3.4 その他

3.4.1 NCシリーズコンパイラ間の移植に関する注意事項

本コンパイラは、弊社製 C コンパイラ「NCxx」と言語仕様レベル(拡張機能を含む)で基本的に互換性を有しています。ただし、以下の点について異なりますのでご注意ください。

(a) near / far のデフォルトの違い

NCシリーズのnear/farのデフォルトは、以下の【表 3.3】通りとなっています。このため、移植時はnear/far指定の調整を必要とする場合があります。

表3.3 NCシリーズの near / far デフォルト

コンパイラ	RAM データ	ROM データ	プログラム
NC100	near (ただし、ポインタ型は far 固定)	far	far 固定
NC308	near (ただし、ポインタ型は far)	far	far 固定
NC30	near	far	far 固定

付録A コマンドオプションリファレンス

付録 A では、本コンパイラのコンパイルドライバの起動方法と起動オプションの機能を説明します。起動オプションの説明では、本コンパイラから起動できるアセンブラとリンケージエディタの起動オプションを併せて記載しています。

A.1 コンパイルドライバの入力書式

```
% nc100△[起動オプション]△<[アセンブリ言語ソースファイル名]△  
[リロケータブルオブジェクトファイル名]△[C 言語ソースファイル名]>
```

% : プロンプトを示します。
< > : 必須項目を示します。
[] : 必要に応じて記述する項目を示します。
△ : スペースを示します。

図A.1 コンパイルドライバの入力書式

```
% nc100 -osample -as100 "-l" -ln100 "-ms" ncrt0.a30 sample.c<RET>
```

<RET> : リターンキーの入力を示します。
※リンク時には必ずスタートアッププログラムを先に指定してください。

図A.2 コンパイルドライバの入力例

A.2 起動オプション

A.2.1 コンパイルドライバの制御に関するオプション

【表A.1】にコンパイルドライバの制御に関する起動オプションを示します。

表A.1 コンパイルドライバの制御オプション

オプション	機能
-c	リロケータブルファイル(拡張子.r30)を作成し、処理を終了します。 ¹
-D 識別子名	識別子を定義します。#define と同じ機能です。
-dsourc (短縮形 -dS)	C 言語ソースリストをコメントとして出力したアセンブリ言語ソースファイル(拡張子".a30")を生成します(アセンブル後も削除しません)。
-dsourc_in_list (短縮形 -dSL)	"-dsourc(-dS)"の機能に加えて、アセンブリ言語リストファイル(.lst)を生成します。
-E	プリプロセスコマンドのみを処理し結果を標準出力に出力します。
-I ディレクトリ名	プリプロセスコマンドの#include で参照するファイルを検索するディレクトリ名を指定します。ディレクトリは最大 256 個まで指定可能です。
-P	プリプロセスコマンドのみを処理しファイル(拡張子.i)を作成します。
-S	アセンブリ言語ソースファイル(拡張子.a30)を作成し、処理を終了します。
-silent	起動時のコピーライトメッセージを出力しません。
-U プリデファインドマクロ名	指定したプリデファインドマクロを未定義にします。

-C

コンパイルドライバの制御

機能 リロケータブルオブジェクトファイル(拡張子.r30)を作成し、処理を終了します。

注意事項 このオプションを選択したときは、アブソリュートモジュールファイル(拡張子.x30)等、ln100で処理した結果出力されるファイルは生成されません。

¹ 起動オプション-c、-E、-P、及び-Sを指定しない場合、nc100はln100まで制御を行い、アブソリュートモジュールファイル(拡張子.x30)まで作成します。

-D 識別子名**コンパイルドライバの制御**

- 機能:** プリプロセスコマンドの**#define**と同じ機能です。
複数の識別子を指定することもできます。
- 書式:** nc100△-D 識別子名=定数△<C言語ソースファイル名>
- ※[=定数]は省略できます。
- 注意事項:** 定義できる識別子の数は、使用しているホストマシンのOSのコマンドラインの最大文字数に制限されることがあります。

-dsource**-dS****コメントオプション**

- 機能:** C言語ソースリストをコメントとして出力した、アセンブリ言語ソースファイル(拡張子".a30")を生成します(アセンブル後も削除しません)。
- 補足説明:**
- (1) **-S** オプションを使用した場合、自動的に、**-dsource** オプションが有効になります。また、生成された ".a30"、".r30"、を削除しません。
 - (2) 本オプションは、アセンブルリストファイルに、C言語ソースリストを出力したいときに使用します。

-dsource_in_list**-dSL****リストファイルオプション**

- 機能:** "-dsource(-dS)"の機能に加えて、アセンブリ言語リストファイル(拡張子".lst")を生成します。

-E**コンパイルドライバの制御**

- 機能** プリプロセスコマンドのみを処理し結果を標準出力に出力します。
- 注意事項** このオプションを選択したときは、アセンブリ言語ソースファイル(拡張子.a30)、リロケータブルオブジェクトファイル(拡張子.r30)、アブソリュートモジュールファイル(拡張子.x30)等、ccom100、as100、及びln100で処理した結果出力されるファイルは生成されません。

-Iディレクトリ名**コンパイルドライバの制御**

- 機能** プリプロセスコマンドの#includeで参照するファイルを検索するディレクトリ名を指定します。最大256個のディレクトリを指定できます。
- 書式** nc100△-I ディレクトリ名△<C言語ソースファイル名>
- 補足説明** -Iオプションに複数のディレクトリdir1とdir2を指定する例を以下に示します。
% nc100 -I dir1 -I dir2 sample.c<RET>
%: プロンプトを示します。
<RET>:リターンキーの入力を示します。
- 注意事項** 指定できるディレクトリ名の数は、使用しているホストマシンのOSのコマンドラインの最大文字数により制限されることがあります。

-P**コンパイルドライバの制御**

- 機能** プリプロセスコマンドのみを起動しファイル(拡張子.i)を作成し処理を終了します。
- 注意事項** (1) このオプションを選択したときは、アセンブリ言語ソースファイル(拡張子.a30)、リロケータブルオブジェクトファイル(拡張子.r30)、アブソリュートモジュールファイル(拡張子.x30)等、ccom100、as100、及びln100で処理した結果出力されるファイルは生成されません。
(2) このオプションにより生成されるファイル(拡張子.i)には、プリプロセッサが生成する#lineは含まれません。#lineを含む結果を得る場合は、-Eオプションを選択し、リダイレクトしてください。

-S**コンパイルドライバの制御**

機 能: アセンブリ言語ソースファイル(拡張子:a30)を作成し、処理を終了します。

注意事項: このオプションを選択したときは、リロケータブルオブジェクトファイル(拡張子:r30)、アブソリュートモジュールファイル(拡張子:x30)等、as100及びn100で処理した結果出力されるファイルは生成されません。

-silent**コンパイルドライバの制御**

機 能: 起動時のコピーライトメッセージを出力しません。

-Uプリデファインドマクロ名**コンパイルドライバの制御**

機 能: プリデファインドマクロ定数を未定義にします。

書 式: nc100△-U プリデファインドマクロ名△<C言語ソースファイル名>

注意事項: 未定義にできるマクロの数は、使用しているホストマシンのOSのコマンドラインの最大文字数により制限されることがあります。
__STDC__、__LINE__、__FILE__、__DATE__、__TIME__は未定義にすることはできません。

A.2.2 出力ファイル指定オプション

【表A.2】に出力するアブソリュートモジュールファイルの名称を指定する起動オプションを示します。

表A.2 出力ファイル指定オプション

オプション	機能
-dir ディレクトリ名	ln100 が生成するファイル(アブソリュートモジュールファイル、マップファイル、等)の出力先ディレクトリを指定できます。
-o ファイル名	ln100 が生成するファイル(アブソリュートモジュールファイル、マップファイル、等)の名称を指定します。また、ディレクトリ名を含んだパス名も指定できます。ファイルの拡張子は必ず省略してください。

-dir ディレクトリ名

出力ファイル指定

機能 出力ファイルの出力先ディレクトリ名を指定できます。

書式 nc100△-dir ディレクトリ名

注意事項: デバッグのためのソースファイル情報は、コンパイラを起動したディレクトリ(カレントディレクトリ)を起点として生成されます。
このため、異なるディレクトリに出力ファイルを生成した場合、デバッグ等にコンパイラを起動したディレクトリを通知する必要があります。

-o ファイル名

出力ファイル指定

機能 ln100が生成するファイル(アブソリュートモジュールファイル、マップファイル、等)の名称を指定します。また、ディレクトリ名を含んだパス名も指定できます。ファイルの拡張子は必ず省略してください。

書式 nc100△-o ファイル名△<C言語ソースファイル名>

A.2.3 バージョン情報及びコマンドライン表示オプション

【表A.3】に使用するクロスツールのバージョン及びコマンドラインを表示する起動オプションを示します。

表A.3 バージョン情報及びコマンドライン表示オプション

オプション	機能
-v	実行中のコマンドプログラム名及びコマンドラインを表示します。
-V	コンパイラの各プログラムの起動時メッセージを表示し、処理を終了します(コンパイル処理は行いません)。

-v

コマンドプログラム名の表示

機能 内部で実行されるコマンドプログラム名を表示しながらコンパイルを実行します。

注意事項 このオプションは、小文字のvを記述します。

-V

バージョン情報の表示

機能 コンパイラの内部で実行される各コマンドプログラムのバージョン情報を表示し、処理を終了します。

補足説明 本オプションはコンパイラが正常にインストールされたか否かを確認するために使用します。コンパイラ内部で実行される各コマンドの正しいバージョン番号はリリースノートに記載しています。リリースノートに記載されているバージョン番号と、本オプションの表示内容が異なる場合、インストールが正常に行われていない可能性があります。

注意事項 (1) このオプションは、大文字のVを記述します。
(2) このオプションを選択した場合、他のオプションはすべて無効になります。

A.2.4 デバッグ用オプション

【表A.4】にC言語レベルデバッグ情報を出力するデバッグの起動オプションを示します。

表A.4 デバッグ用オプション

オプション	機能
-g	デバッグ情報をアセンブリ言語ソースファイル(拡張子.a30)に出力します。これにより、C 言語レベルでバグが可能になります。
-genter	関数呼び出し時に必ず enter 命令を出力します。 デバッガのスタックトレース機能を使用するときは、必ずこのオプションを指定してください。

-g**デバッグ情報の出力**

機能 デバッグ情報をアセンブリ言語ソースファイル(拡張子.a30)に出力します。

注意事項 C言語レベルでバグを行なう場合は、必ず指定してください。本オプションを選択しても、コンパイラの生成コードには影響を与えません。

-genter**enter命令の出力**

機能 関数呼び出し時に必ずenter命令を出力します。

注意事項

- (1) デバッガのスタックトレース機能を使用するときには必ずこのオプションを選択してください。指定しない場合は、正しい結果が得られません。
- (2) このオプションを選択した場合、必要性の有無にかかわらず関数の入口でenter 命令を使用してスタックフレームを構築するコードを生成します。従いまして、ROM容量及び使用するスタック容量が増加する可能性があります。

A.2.5 最適化オプション

【表A.5】にプログラムの実行速度及びROM容量を最小にする最適化を行う起動オプションを示します。

表A.5 最適化オプション

オプション	短縮形	機能
-O[1~5]	なし	レベル毎に速度及び ROM 容量ともに最小にする最大限の最適化を行います。
-O5OA	なし	最適化オプション"-O5"選択時におけるビット操作命令を使用したコード生成を抑制します。
-OR	なし	ROM 容量を重視した最大限の最適化を行います。
-OS	なし	速度を重視した最大限の最適化を行います。
-OR_MAX	-ORM	ROM サイズに効果のある最適化を最大限に行います。
-OS_MAX	-OSM	スピードに効果のある最適化を最大限に行います。
-Ocompare_byte_to_word	-OCBTW	連続した領域のバイト単位の比較をワード単位で行います。
-Oconst	-OC	const 修飾子で宣言した外部変数の参照を定数に置き換える最適化を行います。なお、定数領域は確保されます。
-Ofile_inline	-OFI	全てのインライン関数に対して、インライン展開を行います。
-Oinline_line	-OIL	インライン展開する関数のサイズ（行数）を変更します。
-Oglob_jump	-OGJ	外部分岐の最適化を行います。
-Oglobal_to_inline	-OGTI	グローバル関数を、inline 宣言扱いにします。
-Oloop_unroll[=ループ回数]	-OLU	ループ文を回さずに、ループ回数分コードを展開します。"ループ回数"は省略可能、省略時は最大 5 回のループ文が対象となります。
-Ono_bit	-ONB	ビット操作をまとめる最適化を抑制します。
-Ono_break_source_debug	-ONBSD	ソース行情報に影響する最適化を抑制します。
-Ono_float_const_fold	-ONFCF	浮動小数点の定数畳み込み処理を抑制します。
-Ono_logical_or_combine	-ONLOC	論理 OR をまとめる最適化を抑制します。
-Ono_asmopt	-ONA	アセンブラオプティマイザ"aopt100"による最適化を抑制します。
-Osp_adjust	-OSA	スタック補正コードを取り除く最適化を行います。 これにより ROM 容量を削減することができます。 ただし、使用するスタック量が多くなる可能性があります。
-Ostatic_to_inline	-OSTI	static 宣言された関数を、inline 宣言扱いにします。

主な最適化オプションの効果を【表A.6】に示します

表A.6 最適化オプション効果一覧表

効果	-O	-OR	-OS	-OSA
速度	良	悪	良	良
ROM サイズ	良	良	悪	良
消費スタック	良	同	同	悪

良: 良く(もしくは同じ)なることを意味します。

悪: 悪く(もしくは同じ)なることを意味します。

同: 変化が無いことを意味します。

-O[1-5]**最適化****機能**

速度及びROM容量ともに効果のある最大限の最適化を行います。このオプションは、**-g**オプションと同時に指定することができます。

数字(レベル)を指定しない場合は、**-O3**と同じです。

-O1: **-O3**、**-Ono_bit**、**-Ono_break_source_debug**、**-Ono_float_const_fold**を有効にしたものと同等です。

-O2: **-O1**と同じです。

-O3: 速度及びROM容量ともに効果のある最大限の最適化を行います。

-O4: **-O3**に加え、**-Oconst**を有効にします。

-O5: 共通部分式の最適化(**-OR**同時指定時)、文字列転送比較(**-OS**同時指定時)などを強化した最大限の最適化を行ないます。

但し以下の条件を満たす場合、正常なコードを出力できない可能性があります。

- 異なるポインタ変数が同時に同じタメモリ位置を指す場合。
- それらの変数を同一関数内で使用する場合。

例:

```
int    a = 3;
int    *p = &a;

void   test1(void)
{
    int    b;
    *p = 9;
    a = 10;
    b = *p;          /* 最適化により"*p"を"9"に置き換えてしまう */
    printf( "b = %d (expect b = 10)\n", b );
}
```

実行結果:

```
b = 9 (expect =10)
```

-O[1-5]

最適化

注意事項:

R32C/100のSFR 領域のレジスタへの書き込み、読み出しには、ビット操作命令(BTSTC、BTSTS)を、使用することはできません。

本コンパイラでは、最適化オプション(-O5)を使用した場合、アセンブラコードに対して、ビット操作命令(BTSTC、BTSTS)を生成する場合があります。

以下の例のような記述を行い、最適化オプション(-O5)を使用してコンパイルした場合、割り込み要求ビットの判定が正常に行われず意図しない動作を行います。

```
#pragma ADDRESS ta0ic_addr 006CH /* タイマ A0 割り込み制御レジスタ */

struct {
  char ilvl :3;
  char ir   :1; /* 割り込み要求ビット */
  char dmy  :4;
} ta0ic;

void wait_until_IR_is_ON(void)
{
    while (ta0ic.ir == 0) /* 1 になるまで待つ */
    {
        ;
    }
    ta0ic.ir = 0; /* 1 になったら 0 に戻す */
}
```

SFR領域に対してビット操作命令(BTSTC、BTSTS)が出力されていることが確認されたら、以下のような対策を行った上でコンパイルしてください。いずれの場合も、生成されたコードに問題が無いことを必ず確認してください。

- "-O5"以外の最適化オプションを使用する。"-O5"の最適化オプションを使用する場合は、"-O5OA"と併用して下さい。
- asm 関数を使用してプログラム中に直接命令を記述する。

-O50A**最適化**

機能 最適化オプション"-O5"選択時におけるビット操作命令を使用したコード生成を抑制します。

-OR**最適化**

機能 速度は低下する場合がありますが、ROM容量を重視した最適化を行います。このオプションは、-gオプション、-Oオプションと同時に指定することができます。

注意事項 このオプションを使用した場合、ソース行情報の一部を変更する最適化を行なう可能性があります。このため、デバッグ時に動作が異なって見える場合があります。

ソース行情報を変更したくない場合、-Ono_break_source_debug(-ONBSD)オプションを使用して最適化を抑制してください。

-OR_MAX**-ORM****最適化**

機能 ROMサイズを優先する最適化を行います。
本オプションが指定された場合は、-O5、-O50A、-OGJ、-OR、-fD32、-fNA、-fUFが有効となります。

-OS**最適化**

機能 ROM容量は増大する場合がありますが、速度を重視した最適化を行います。
このオプションは、-gオプション、-Oオプションと同時に指定することができます。

-OS_MAX**-OSM**

最適化

機能

スピードを優先する最適化を行います。

本オプションが指定された場合は、**-O4**、**-OGJ**、**-OSTI**、**-OGTI**、**-OS**、**-OSA**、**-OLU=10**、**-fD32**、**-fUF** が有効となります。

-Ocompare_byte_to_word**-OCBTW**

最適化

機能

連続した領域のバイト単位の比較をワードで行います。

注意事項

-O[1-5](もしくは、**-OR**、**-OR_MAX(-ORM)**、**-OS**、**-OS_MAX(-OSM)**)オプションを選択したときのみ効果があります。

-Oconst**-OC**

最適化

機能

const修飾子で宣言した、変数の参照を定数に置き換える最適化を行います。なお、定数領域は確保されません。また**-O4**オプション以上の指定時にも有効になります。

補足説明

以下の条件を同時に満たした場合に最適化を行います。

- (1) ビットフィールドおよび、共用体を除く変数
- (2) **const**修飾子を指定し、かつ **volatile** 指定をしていない変数
- (3) 同一のC言語ソースファイル中で初期化を記述している外部変数
- (4) 定数または、**const** 修飾子を指定された変数で初期化している変数

-Ofile_inline [=インライン展開ファイル(, ...)]**-OFI [=インライン展開ファイル(, ...)]****インライン展開****機 能**

- 全てのインライン関数に対して、インライン展開を行います。
- 参照されない **static**関数のコード生成を抑止します。
- インライン展開ファイルが指定された場合、ファイル間にまたがったグローバル関数に対してインライン展開を実施します。
- ファイル間にまたがったグローバル関数のインライン展開は、展開される関数（{から}まで）のサイズがコメント行を含む150行以内の関数に対してインライン展開を実施します。なおサイズについてオプション"-Oinline_line"で変更することができます。

補足説明:

インライン関数の宣言とその実体定義は、インライン関数の宣言をした後に、インライン関数の実体定義を行わなければ成りませんが、本オプションを使用する事により、インライン関数の宣言を行う前に、インライン関数の実体定義を行う事ができます。

- オプション"-Ofile_inline"指定によりインライン展開される記述例を示します。

```
extern int i;
inline int func(void)
{
    return i++;
}

void main(void)
{
    int s;

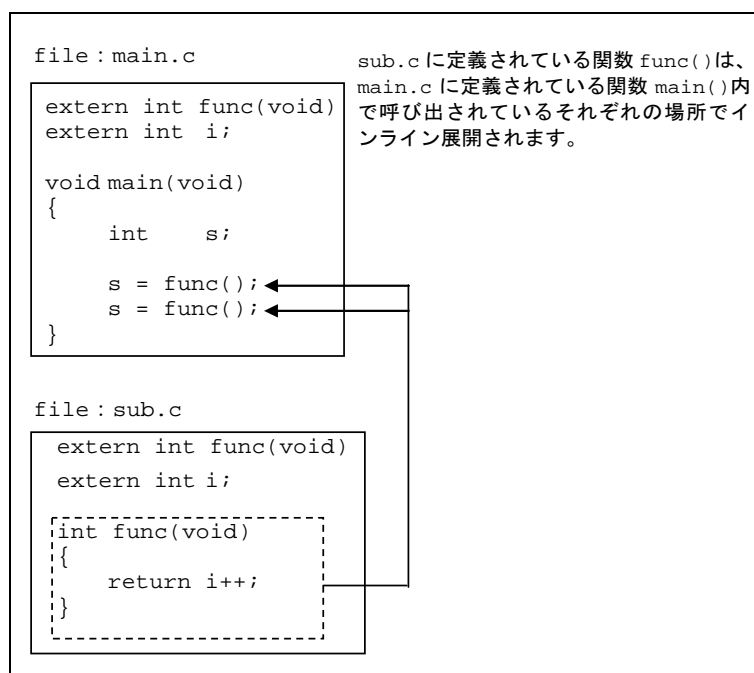
    s = func();
    s = func();
}

```

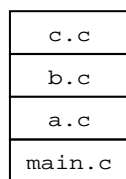
関数 func()は、関数 main()内で呼び出されているそれぞれの場所でインライン展開されます。

また、インライン展開ファイルを指定した場合、ファイル間にまたがったグローバル関数をインライン展開することができます。

- オプション"-Ofile_inline=インライン展開ファイル"指定によりインライン展開される記述例を示します。下記例では、ソースファイルmain.cに対して、オプション"-Ofile_inline=sub.c"指定によりインライン展開される記述例となります。

-Ofile_inline [=インライン展開ファイル(, ...)]**-OFI [=インライン展開ファイル(, ...)]****インライン展開****注意事項:**

- (1) インライン関数の宣言と、インライン関数の実体定義は、同一ファイル内に記述してください。
- (2) インライン関数の引数には、構造体や共用体を使用する事はできません。これらを使用した場合、コンパイルエラーとなります。
- (3) インライン関数の間接呼び出しをすることはできません。間接呼び出しの記述を行った場合、コンパイルエラーとなります。
- (4) インライン関数の再帰呼び出しをすることはできません。再帰呼び出しの記述を行った場合、コンパイルエラーとなります。
- (5) インライン展開ファイルが複数指定された場合は、指定されたファイルの順番でインライン展開を実施します。“main.c -Ofile_inline=a.c,b.c,c.c” と指定された場合は、下記ファイル構成として処理します。



矢印の方向に対して処理を実施するため、a.c, b.c および c.c では、前方参照する関数のインライン展開は実施されません。

- (6) programセクションのみ対象になります。“#pragma SECTION”でセクション名を変更した場合、ファイル間インライン展開は実施しません。
- (7) “#pragma __ASMMACRO”で定義された“(アンダーバー)”で始まるstatic関数(asmmacro.h, string.hに定義)は、インライン展開の対象になります。

-Oinline_line =インライン展開行**-OIL=インライン展開行
インライン展開**

機能 オプション"-Ofile_inline"、"-Oglobal_to_inline"および"-Ostatic_to_inline"指定によりインライン展開する関数のサイズ（行数）を変更します。
本オプション省略時は、インライン展開される関数の '{ から }'までのサイズがコメント行を含む150行以内の関数が対象となります。

-Oglb_jmp**-OGJ
最適化**

機能 分岐命令に関する外部参照の最適化を行います。

-Oglobal_to_inline**-OGTI
グローバル関数をinline関数扱いにする**

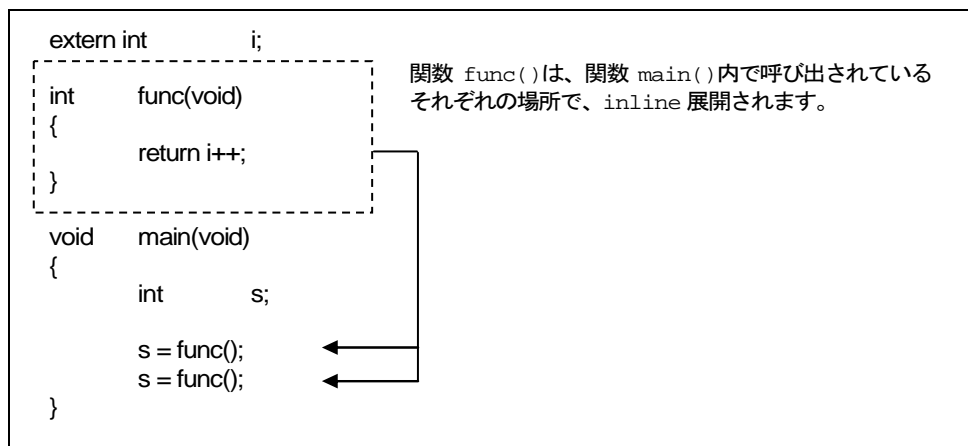
機能 グローバル関数を、**inline**宣言されている関数(**inline**関数)として扱い、**inline**展開したアセンブルコードを生成します。

補足説明: 以下の条件を満たした場合、グローバル関数を**inline**関数として扱い、**inline**展開したアセンブルコードを生成します。

- (1) 関数呼び出しの前に、実体が記述されているグローバル関数を対象とします。
 - 関数の呼び出しと、その関数の実体が、同じソースファイル内に記述されていなければなりません。
 - "-Ofile_inline"オプションを選択した場合は、本条件を無視してください。
- (2) 対象となるグローバル関数に対して、プログラム中でアドレス取得を行っていない場合。
- (3) 対象となるグローバル関数を再帰呼び出ししていない場合。

-Oglobal_to_inline**-OGTI****グローバル関数をinline関数扱いにする**

以下に、`inline`展開されるグローバル関数の記述例を示します。

**注意事項:**

- (1) `inline`関数扱いになったグローバル関数の実体の記述に対するアセンブラコードは、常に生成されます。
- (2) 関数を強制的に `inline`関数扱いする場合は、`inline`宣言してください。
- (3) インライン展開される関数は、関数の '{ から }' までのサイズがコメント行を含む150行以内の関数が対象となります。なおサイズについてオプション"`-Oinline_line`"で変更可能になります。

-Oloop_unroll[=ループ回数]**-OLU[=ループ回数]****ループの展開****機能**

ループ文を回さずにループ回数分コードを展開します。"ループ回数"は省略可能、省略時は最大5回のループ文が対象となります。

補足説明:

実行回数が明確であるfor文 に対してのみ展開したコードを出力します。

for展開を行う際に対象とする for の回転数の上限を指定します。デフォルトでは、5回転以下のfor文が対象となります。

注意事項:

for文を展開するため、ROMサイズは増加します。

-Ono_bit**-ONB****最適化の抑止****機能**

ビット操作をまとめる最適化を抑止します。

補足説明:

-O[3~5](もしくは-OR、-OS)オプションを選択した場合は、同じメモリ領域に配置されたビットフィールドに対して連続に定数を代入する操作を1つの操作にまとめる最適化を行います。入出力等のビットフィールドにおいて連続するビット操作に順序がある場合この最適化は望ましくありませんので、本オプションを使用して最適化を抑止してください。

注意事項:

-O[3~5](もしくは-OR、-OS)オプションを選択したときのみ効果があります。

-Ono_break_source_debug**-ONBSD****最適化の抑止****機能**

ソース行情報に影響する最適化を抑止します。

補足説明:

-OR、-O[3~5] オプション指定には、ソース行情報に影響する最適化を行う可能性があります。本オプションは、ソース行情報に影響する最適化を抑止する場合に使用します。

注意事項:

-OR、-O[3~5] オプションを選択したときのみ効果があります。

-Ono_float_const_fold**-ONFCF****最適化の抑止****機能**

浮動小数点の定数畳み込み処理を抑止します。

補足説明:

本コンパイラでは、デフォルトで定数の畳み込み処理を行います。定数の畳み込み処理の例を以下に示します。

最適化前:

```
(val/1000e250)*50.0
```

最適化後:

```
val/20e250
```

この場合に、浮動小数点のダイナミックレンジ全体を使用したアプリケーションでは、計算順序を換えることにより計算結果が異なる場合があります。本オプションは、浮動小数点における定数の畳み込みを抑止し、Cソースに記述した計算順序を保証します。

-Ono_logical_or_combine**-ONLOC****最適化の抑止**

機 能 論理ORをまとめる最適化を抑止します。

補足説明 下記の例のように、**-O3** 以上、**-OR**、**-OS** のいずれかを指定してコンパイルした場合、論理ORをまとめる最適化を行います。

例:

```
if( a & 0x01 || a & 0x02 || a & 0x04 )
```

↓ (最適化)

```
if( a & 0x07 )
```

この場合、変数a に対して最大3回の参照が行われますが、最適化後は1回の参照になります。しかし、変数aが、I/Oなどの参照に意味がある場合、正しい動作が行われない可能性があります。この場合は、本オプションを選択して論理ORをまとめる最適化を抑止してください。なお、変数に **volatile**宣言がされている場合は、論理ORをまとめる最適化は行われません。

-Ono_asmopt**-ONA****アセンブラオプティマイザの抑止**

機 能 アセンブラオプティマイザ“aopt100”による最適化を抑止します。

-Osp_adjust**-OSA****スタック補正コードをまとめる**

機能 関数呼び出し後のスタック補正コードをまとめる最適化を行います。
なお、本オプションは-O[1~5]と併用してください。

補足説明: 通常は関数呼び出し毎に、関数の引数の領域を解放するために、スタックポインタを補正する処理をします。本オプションを使用することにより、このスタックポインタの補正を関数の呼出し毎ではなく、まとめて行うようにします。

例:

下記の場合、func1()、 func2() それぞれの呼び出し毎にスタックポインタの補正 (2回の補正)が行われるが、本オプションを使用した場合は1回の補正となる。

```
char    func1(char, char, char);
char    func2(char, char, char);

void    main( void ) {
    char    i = 1;
    char    j = 2;
    char    k=3;
    char    l, m;

    l = func1( i, j, k);
    m = func2( i, j, k );
}
```

注意事項: オプション-Osp_adjustによりROM容量を削減し、かつ速度を向上することができます。ただし、使用するスタック量が多くなる可能性があります。

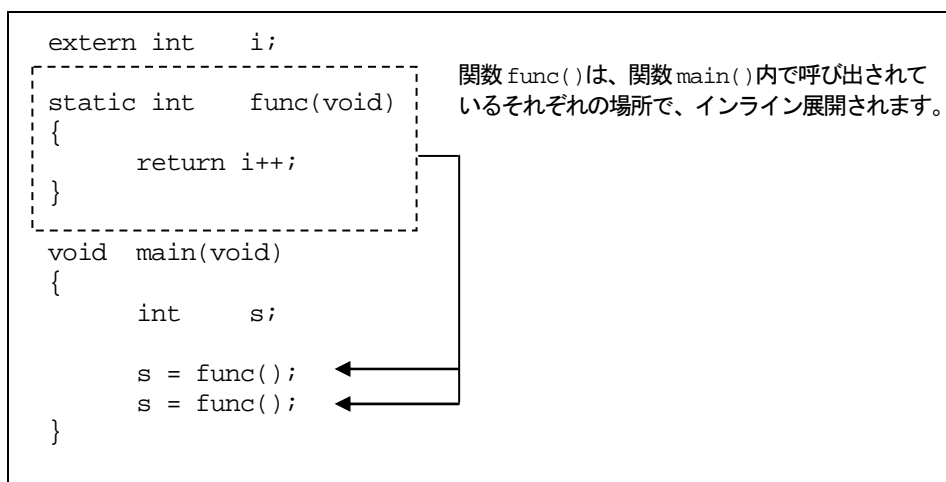
-Ostatic_to_inline**-OSTI****static関数をinline関数扱いにする**

機能 static宣言された関数(static関数)を、inline宣言されている関数(inline関数)として扱い、インライン展開したアセンブルコードを生成します。

補足説明: 以下の条件を満たした場合、static関数をinline関数として扱い、インライン展開したアセンブルコードを生成します。

- (1) 関数呼び出しの前に、実体が記述されているstatic関数を対象とします。
 - 関数の呼び出しと、その関数の実体が、同じソースファイル内に記述されていなければなりません。
 - オプション"-Ofile_inline"を選択した場合は、本条件を無視してください。
- (2) 対象となるstatic関数に対して、プログラム中でアドレス取得を行っていない場合。
- (3) 対象となるstatic関数を再帰呼び出ししていない場合。

以下に、インライン展開されるstatic関数の記述例を示します。



- 注意事項:**
- (1) inline関数扱いになったstatic関数の実体の記述に対するアセンブラコードは、常に生成されます。ただし、オプション"-Ofile_inline"と併用した場合は生成されません。
 - (2) 関数を強制的に inline関数扱いする場合は、inline宣言してください。
 - (3) インライン展開される関数は、関数の '{ から }' までのサイズがコメント行を含む150行以内の関数が対象となります。なおサイズについてオプション"-Oinline_line"で変更可能になります。

A.2.6 生成コード変更オプション

【表A.7】にnc100が生成するアセンブリ言語を制御する起動オプションを示します。

表A.7 生成コード変更オプション

オプション	短縮形	機能
-fansi	なし	"-fnot_reserve_far_and_near"、"-fnot_reserve_asm"、"-fnot_reserve_inline"及び"-fextend_to_int"を有効にします。
-fconst_not_ROM	-fCNR	const で指定した型をROM データとして扱いません。
-fdouble_32	-fD32	double 型を float 型として処理します。
-fenable_register	-fER	レジスタ記憶クラスを有効にします。
-fextend_to_int	-fETI	char型およびshort型データをint型に拡張し演算します(ANSI規格で定められた拡張を行います)。 ²
-ffar_RAM	-fFRAM	RAM データのデフォルト属性を far にします。
-finfo	なし	"Call Walker"、"Map Viewer"に必要な情報を出力します。
-fint_16	-fI16	int 型を 16 ビット幅として扱います。
-fJSRW	なし	関数呼び出しの命令のデフォルトを JSR.W 命令に変更します。
-fnear_ROM	-fNROM	ROM データのデフォルト属性を near にします。
-fno_align	-fNA	関数の先頭アドレスのアライメントを行いません。
-fno_switch_table	-fNST	switch 文に対し、比較を行ってから分岐するコードを生成します。
-fnot_address_volatile	-fNAV	#pragma ADDRESS で指定した変数を volatile で指定した変数とみなしません。
-fnot_reserve_asm	-fNRA	asm を予約語にしません("_asm"のみ有効になります)。
-fnot_reserve_far_and_near	-fNRVAN	far、near を予約語にしません(_far、_near のみ有効になります)。
-fnot_reserve_inline	-fNRI	inline を予約語にしません(_inline のみ予約語となります)。
-fsigned_char	-fSC	符号指定のない char 型を signed char 型として扱います。
-fswitch_other_section	-fSOS	switch 文に対するテーブルジャンプをプログラムセクションとは別セクションに出力します。
-fuse_FPU	-fUF	FPU 命令を出力します。

² ANSI 規格では、(unsigned)char型、signed char型、short型およびunsigned short型データを評価する時に必ずint 型に拡張します。

これはchar型の演算、例えば、「c1=c2*2/c3;」を行うときに演算の途中でchar型をオーバーフローし、結果が予期せぬ値になるのを防ぐためです。

-fansi

生成コードの変更

機能 以下に示す起動オプションをすべて有効にします。

-fnot_reserve_asm	asm を予約語として扱いません。
-fnot_reserve_far_and_near	far、near を予約語として扱いません。
-fnot_reserve_inline	inline を予約語として扱いません。
-fextend_to_int	char 型データを int 型に拡張して演算を行います。

補足説明: このオプションを選択することにより、ANSI規格に基づいたコード生成を行います。

-fconst_not_ROM**-fCNR**

生成コードの変更

機能 const修飾子で指定した型をROMデータとして扱いません。

補足説明: フォルトでは、const指定したデータはROM領域に配置されます。

```
int const array[10] = { 1,2,3,4,5,6,7,8,9,10 };
```

上記の場合、配列「array」は、ROM領域に配置されます。本オプションを指示することにより、この「array」をRAM領域に配置することができます。

通常の用途では、本オプションを使用する必要はありません。

-fdouble_32**-fD32**

生成コードの変更

機能 double 型をfloat 型として処理します。

補足説明: (1) 本オプションを選択した場合は、必ず、関数のプロトタイプ宣言を行なってください。プロトタイプ宣言がない場合は、不正なコードを生成する場合があります。
 (2) 本オプションを選択した場合のdouble 型に対するデバッグ情報はfloat 型として扱われます。このため、エミュレータデバッガ、シミュレータデバッガのCウォッチウインドウ、グローバルウインドウ等ではfloat型として表示されます。

-fenable_register**-fER**

生成コードの変更

機能 register記憶クラスを指定した変数をレジスタに割り当てます。

補足説明: 「auto変数をレジスタに割り当てる最適化」を行った場合、必ずしも最適解を得られるとは限りません。本オプションは上記状況下において、プログラム上でレジスタ割り当てを指示する事により効率を高める手段として用意しています。

本オプションを選択することにより、**register**指定された以下の変数を強制的にレジスタに割り当てます。

- 整数型変数
- 浮動小数点型変数
- ポインタ変数

注意事項: むやみに**register**指定を行うと逆に効率を低下させる場合があります。必ず生成されたアセンブリ言語を確認の上、使用してください。

-fextend_to_int**-fETI**

生成コードの変更

機能 (unsigned)char型、signed char型、short型およびunsigned short型データをint型に拡張し演算を行います(ANSI規格で定められた拡張を行います)。

補足説明: ANSI規格では、(unsigned)char型、signed char型、short型およびunsigned short型データを評価する時に必ずint型に拡張します。これはchar型の演算、例えば、 $c1 = c2 * 2 / c3$;を行うときに演算の途中でchar型をオーバーフローし、結果が予期せぬ値になるのを防ぐためです。

```
void    main(void)
{
    char    c1;
    char    c2 = 200;
    char    c3 = 2;

    c1 = c2 * 2 / c3;
}
```

この場合「 $c2 * 2$ 」の演算でchar型をオーバーフローし、正しい結果を求められません。

本オプションを選択することにより、正しい結果を求めることができます。

デフォルトの設定をint型への拡張を行わないようにしているのは、ROM効率を少しでも良くするためです。

-ffar_RAM**-fFRAM**

生成コードの変更

機能 RAMデータのデフォルト属性をfar属性にします。

補足説明: RAMデータ(変数)は、デフォルトでnear領域に配置されます。near領域(64K バイトの領域)の外にRAMデータを配置する場合に本オプションを使用します。

-finfo

生成コードの変更

機能 "Call Walker"、"Map Viewer"に必要な情報を出力します。

補足説明: "Call Walker"、"Map Viewer"を使用する時は、このオプションの選択により生成されたアプソリュートファイル(.x30)が必要です。

-fint_16**-fi16**

生成コードの変更

機能 int型を16ビット幅として扱います。

補足説明: 本オプションを使用する場合は、標準ライブラリとして、nc100lib.libの代わりにnc100i16.libをリンクする必要があります。コンパイルドライバから本オプションを指定してリンクまでを行った場合には、リンクするライブラリの変更は自動的に行われます。

本オプションを指定しない場合のint型のサイズは、32ビットです。

また、本オプションと“-fuse_FPU(-fiUF)”を組み合わせる場合は、標準ライブラリとして、nc100lib.libの代わりにnc100i16fpu.libをリンクする必要があります。コンパイルドライバから本オプションと“-fuse_FPU(-fiUF)”を組み合わせるリンクまでを行った場合には、リンクするライブラリの変更は自動的に行われます。

-fJSRW**生成コードの変更**

- 機能** 関数呼び出し命令のデフォルトをJSR.W命令に変更します。
- 補足説明:** デフォルトでは、ソースファイルの外で定義された関数を呼び出す時にJSR.A命令を用いて呼び出します。本オプションを選択した場合、JSR.A命令をJSR.W命令に変更することができるため、生成コードサイズを圧縮することができます。
本オプションは、プログラムサイズが32Kバイト以内、もしくはROM圧縮をしたい場合に有効です。
- 注意事項:** 呼び出し位置から前後32Kバイトを超える位置にある関数をJSR.W命令で呼び出すと、リンク時にエラーが発生します(#pragma JSRAを組み合わせることにより、このエラーを回避することができます)。

-fnear_ROM**-fNROM****生成コードの変更**

- 機能** ROMデータのデフォルト属性をnear属性にします。
- 補足説明:** ROMデータ(const指定された変数等)は、デフォルトでfar領域に配置されます。本オプションを選択することにより、ROMデータをnear領域に配置することができます。

-fno_align**-fNA****生成コードの変更**

- 機能** 関数の先頭アドレスのアライメントを行いません。

-fno_switch_table**-fNST**

生成コードの変更

- 機能** **switch**文に対して、ジャンプテーブルを用いたコードを生成せず、比較を行ってから分岐するコードを生成します。
- 補足説明:** 本オプションを選択しない場合は、コードサイズがより小さくなる場合のみ、ジャンプテーブルを用いたコードを生成します。
- 注意事項:** 1関数のコードサイズが32Kバイトを超えるような大きな関数では、**switch**文に対してジャンプテーブルを用いたコードを生成すると、正しいアドレスに分岐できない場合があります。この場合、必ず本オプションを指定してください。
 なお、本オプションを指定せずに、正しく分岐できないコードを生成した場合に、コンパイラ、アセンブラ、およびリンカでは正しく分岐できないことに関する警告およびエラーメッセージは出力されませんので、ご注意ください。

-fnot_address_volatile**-fNAV**

生成コードの変更

- 機能** **#pragma ADDRESS**又は**#pragma EQU**で指定した大域変数又は関数外に宣言した**static**変数を、**volatile**で指定された変数として扱いません。
- 補足説明:** I/O変数をRAM上に在る変数と同じ最適化を行うと、期待した動作をしない場合があります。これは、I/O変数に**volatile**指定をすることにより避けることができます。
"#pragma ADDRESS"又は**"#pragma EQU"**は、通常、I/O変数に対して使用するため、**volatile**指定が無くても、**volatile**指定がされているものとして処理されます。
 本オプションは、この処理を抑止します
- 注意事項:** 通常の用途では、本オプションを使用する必要はありません。

-fnot_reserve_asm**-fNRA**

生成コードの変更

- 機能** **asm**を予約語として扱いません。
- 補足説明:** 同じ機能の**_asm**は予約語として扱われます。

-fnot_reserve_far_and_near**-fNRFAN**

生成コードの変更

機能: far、nearを予約語として扱いません。

補足説明: 同じ機能の_far、_nearは予約語として扱われます。

-fnot_reserve_inline**-fNRI**

生成コードの変更

機能: inlineを予約語として扱いません。

補足説明: 同じ機能の_inlineは予約語として扱われます。

-fsigned_char**-fSC**

生成コードの変更

機能: 符号指定のないchar型をsigned char型として扱います。

-fswitch_other_section**-fSOS**

生成コードの変更

機能: switch文に対するテーブルコードをプログラムセクションとは別のセクションに出力します。

補足説明: セクション名は、switch_tableです

注意事項: 本オプションは通常、使用する必要はありません。

-fuse_FPU**-fUF**

生成コードの変更

機 能 FPU命令を出力します。

補足説明: 本オプションを使用する場合は、標準ライブラリとして、`nc100lib.lib`の代わりに`nc100fpu.lib`をリンクする必要があります。コンパイルドライバから本オプションを指定してリンクまでを行った場合には、リンクするライブラリの変更は自動的に行われます。

また、本オプションと“`-fint_16(-fl16)`”を組み合わせる場合は、標準ライブラリとして、`nc100lib.lib`の代わりに`nc100i16fpu.lib`をリンクする必要があります。コンパイルドライバから本オプションと“`-fint_16(-fl16)`”を組み合わせるリンクまでを行った場合には、リンクするライブラリの変更は自動的に行われます。

A.2.7 ライブラリ指定オプション

【表A.8】にライブラリファイルを指定する起動オプションを示します。

表A.8 ライブラリ指定オプション

オプション	機能
-lライブラリファイル名	リンク時に使用するライブラリを指定します。

-lライブラリファイル名

機 能: ln100がリンク時に使用するライブラリファイル名を指定します。ファイルの拡張子は省略可能です。

書 式: nc100△-lファイル名△<C言語ソースファイル名>

- 注意事項:**
- (1) ファイル指定では拡張子を省略することができます。拡張子を省略した場合のファイルの拡張子は".lib"として処理されます。
 - (2) ファイルの拡張子を指定する場合は、必ず".lib"を指定してください。
 - (3) NC100は環境変数LIB100で指定されたディレクトリ内にあるライブラリ"nc100lib.lib"をデフォルトでリンクします。なおコンパイルオプションの指定により、リンクされるライブラリファイルは次の通りです。

コンパイルオプション "-fint_16"	コンパイルオプション "-fuse_FPU"	参照ライブラリ
指定なし	指定なし	nc100lib.lib
指定あり	指定なし	nc100i16.lib
指定なし	指定あり	nc100fpu.lib
指定あり	指定あり	nc100i16fpu.lib

- (4) 複数のライブラリを指定した場合、"nc100lib.lib"を参照する優先順位は最も低くなります。

A.2.8 警告オプション

【表A.9】にnc100の言語仕様に関する記述の間違いに対して警告(ワーニングメッセージ)を出力する起動オプションを示します。

表A.9 警告オプション

オプション	短縮形	機能
-Wall	なし	検出可能な警告("-Wlarge_to_small"、"-Wno_used_argument"で出力される警告を除く)をすべて表示します。
-Wccom_max_warnings = ワーニング回数	-WCMW	ccom100の出力するワーニングの回数の上限を指定できます。
-Werror_file<ファイル名>	-WEF	エラーメッセージを指定したファイルに出力します。
-Wignore_near_pointer	-WINP	near ポインタを far ポインタとして扱う際の警告を抑止します。
-Wlarge_to_small	-WLTS	大きいサイズから小さいサイズへの暗黙の代入に対して、ワーニングを出力します。
-Wmake_tagfile	-WMT	error および warning が発生した場合はファイル毎にタグファイルを出力します。
-Wmultiple_tentative_definitions	-WMTD	同一変数に対して複数の仮定義がある場合に警告します。
-Wnesting_comment	-WNC	コメント中に"/"を記述した場合に警告を出します。
-Wno_stop	-WNS	エラーが発生してもコンパイル作業を停止しません。
-Wno_used_argument	-WNUA	引数を持つ関数を定義した場合に、使用していない引数に対してワーニングを出力します。
-Wno_used_function	-WNUF	リンク時に未使用のグローバル関数を表示します。
-Wno_used_static_function	-WNUSF	コード生成が不要な static 関数名を表示します。
-Wno_warning_stdlib	-WNWS	"-Wnon_prototype"指定時や"-Wall"指定時に本オプションを指定すると、「プロトタイプ宣言されていない標準ライブラリに対する警告」を抑止します。
-Wnon_prototype	-WNP	プロトタイプ宣言されていない関数を使用した場合、警告を出します。
-Wstdout	なし	エラーメッセージをホストマシンの標準出力(stdout)に出力します。
-Wstop_at_link	-WSAL	リンク時にワーニングが発生した場合、アブソリュートモジュールファイルの生成を抑止します。
-Wstop_at_warning	-WSAW	ワーニング発生時にコンパイル処理を停止します。
-Wundefined_macro	-WUM	#if の中で未定義のマクロを使用した場合に警告します。
-Wuninitialize_variable	-WUV	初期化されていない auto 変数に対してワーニングを出力します。
-Wunknown_pragma	-WUP	サポートしていない #pragma を使用した場合、警告を出します。

-Wall**警告オプション**

- 機能** 検出可能な警告をすべて表示します。
- 補足説明:**
- (1) "-Wlarge_to_small(-WLTS)"、および"-Wno_used_argument(-WNUA)"、"-Wno_used_static_function(-WNUSF)"を使用した場合の警告は除きます。
 - (2) オプション"-Wnon_prototype(-WNP)"、"-Wunknown_pragma(-WUP)"、"-Wnesting_comment(-WNC)"、"-Wuninitialize_variable(-WUV)"と同等の警告を表示します。
 - (3) 以下の場合にも警告を表示します。
 - if文、for文や、&&、|| 演算子の比較文に代入演算子"="を使用した場合。
 - 代入演算子"="を間違って"=="と記述した場合。
 - 古い形式の関数定義を行った場合。
- 注意事項:** これらの警告は、コンパイラの判断で誤った記述と推測できる範囲で検出しています。このためすべての誤りを警告できるとは限りません。

-Wccom_max_warnings= ワーニング回数**-WCMW=** ワーニング回数**警告オプション**

- 機能** コンパイラ本体の出力するワーニングの回数の上限を指定できます。
- 補足説明:** デフォルトでは、ワーニングの出力には上限がありません。本オプションは、多量のワーニング出力により、画面がスクロールするのを調節する場合等に使用します。
- 注意事項:** ワーニングの出力の上限回数は、0回以上で指定してください。また、指定回数の省略はできません。0回を指定するとワーニング出力を完全に抑止します。

-Werror_file <ファイル名>**警告オプション**

機能 エラーメッセージを指定したファイルに出力します。

書式 nc100△-Werror_file△<エラー出力ファイル名>

注意事項 ファイルに出力されるエラーメッセージの出力フォーマットは、ディスプレイに表示されるエラーメッセージとは異なり、一部のエディタの持つ「タグジャンプ機能」に適したフォーマットで出力されます。

-Wignore_near_pointer**-WINP****警告オプション**

機能 nearポインタをfarポインタとして扱う際の警告を抑制します。

補足説明 本コンパイラでは、ポインタの属性はfar(32ビット)固定です。デフォルトでは、ポインタに対するnear修飾は、警告した上で無視されます。

本オプションを使用すると、ポインタのnear修飾を無視したという警告を抑制します。

-Wlarge_to_small**-WLTS****警告オプション**

- 機能** 大きいサイズから、小さいサイズへの暗黙の代入に対して、ワーニングを出力します。
- 補足説明:** 各型の負数の境界値では、型に収まる数値であっても、警告を出力する場合があります。これは言語規約上、負数は単項演算子 (-) と整数が結合したものであるためです。例えば「-32768」は、signed int 型に収まる値ですが、「-」と"32768"に分解すると、"32768"は signed int 型に収まらないため、"signed long型"になります。従って即値"-32768"は、signed long型です。このため、「 int i = -32768; 」のような記述に対して、警告が出力されます。
- 注意事項:** 本オプションは、多量のワーニングを出力するため、以下の型変換のみワーニング出力を抑制しています。
- char型変数 から char型変数への代入
 - 即値のchar型変数 への代入
 - 即値のfloat型変数への代入

-Wmake_tagfile**-WMT****警告オプション**

- 機能** error および warning が発生した場合に、ファイル単位で、タグファイル に生成メッセージの内容を出力します。
- 補足説明:** 本オプションを選択した場合に"-Werror_file <ファイル名> (-WEF)"を同時に指定した場合は、エラーとなります。

-Wmultiple_tentative_definitions**-WMTD****警告オプション**

- 機能** 同一変数に対して、複数の仮定義がある場合に警告します。
- 補足説明:** 関数外で、初期化子を使わず、かつ、記憶クラス指定子なしか又はstaticで変数を宣言する場合、この宣言を仮定義といいます。本オプションを使用すると、このような宣言が複数回行われた場合に警告します。

-Wnesting_comment**-WNC****警告オプション**

機能 コメント内に"/"を記述している場合に警告を発生します。

補足説明 本オプションを使用することにより、コメントのネストを検出することができます。

-Wno_stop**-WNS****警告オプション**

機能 エラーが発生してもコンパイル作業を停止しません。

補足説明 コンパイラは関数単位でコンパイルします。コンパイル中にエラーが発生すると、デフォルトでは、次の関数のコンパイルを行いません。

また、エラーが原因で別のエラーを引き起こすことがあり、エラーが多いとコンパイルを停止します。

本オプションを使用することにより、可能な限りコンパイルを続けます。

注意事項 記述によるエラーが原因で**System Error**が発生する場合があります。その場合は、本オプションを使用している場合であってもコンパイル作業が停止します。

-Wno_used_argument**-WNUA****警告オプション**

機能 引数を持つ関数を定義した場合、使用していない引数に対してワーニングを出力します。

-Wno_used_function**-WNUF****警告オプション**

機能 未使用のグローバル関数を、リンク時に表示します。

注意事項 本オプションを選択する場合は、必ず"/finfo"オプションも同時に指定してください。

-Wno_used_static_function**-WNUSF****警告オプション**

機能 コード生成が不要な**static**関数名を表示します。条件は下記に示すいずれかの場合です。

- **static**関数がファイルのどこからも参照されない。
- **"-Ostatic_to_inline(-OSTI)"**オプションにより、**static**関数が**inline**化される。

注意事項 下記に示すように配列の初期化子に関数名を記述した場合は、プログラム動作時に参照されない関数であってもコンパイラは参照されるものとして処理します。下記の例では、関数**f4**と**f5**は参照されませんが、コンパイラはこれらの関数を参照されるものとして処理します。

```
例:  
void      (*a[5])(void) = {f1,f2,f3,f4,f5};  
  
          for(i = 0; i < 3; i++) (*a[i])();
```

-Wno_warning_stdlib**-WNWS****警告オプション**

機能 **"-Wnon_prototype"**や、**"-Wall"**と同時に本オプションを選択すると、「プロトタイプ宣言されていない標準ライブラリに対する警告」を抑制します。

-Wnon_prototype**-WNP****警告オプション**

機能 前もってプロトタイプ宣言がされていない関数を使用した場合、または関数のプロトタイプ宣言を行っていない場合に警告を出します。

補足説明: プロトタイプ宣言を行うことにより、関数引数をレジスタ渡しにすることができます。レジスタ渡しにすることにより、速度向上、コードサイズ削減が期待できます。
また、プロトタイプ宣言を行うことにより、コンパイラが関数の引数を検査するようになります。このため、プログラムの信頼性向上を期待できます。
したがって、本オプションは、常に使用することを推奨します。

-Wstdout**警告オプション**

機能 エラーメッセージをホストマシンの標準出力(stdout)に出力します。

補足説明: 本オプションは、Windows版(パソコン版)においてエラー出力等をリダイレクトを用いて、ファイルに保存する場合に使用します。

注意事項: コンパイルドライバから呼び出されるas100、ln100のエラー出力は、本オプションに関係なく標準出力に出力されます。

-Wstop_at_link**-WSAL****警告オプション**

機能 リンク時にワーニングが発生した場合、リンクを停止し、アブソリュートモジュールファイルの生成を抑制します。また、戻り値"10"をホストOSに返します。

-Wstop_at_warning**-WSAW****警告オプション**

機能 コンパイル時にワーニングが発生した場合、コンパイルを停止し、コンパイラの終了コード "10"を返します。

補足説明: デフォルトでは、コンパイル時にワーニングが発生した場合、コンパイルの終了コードは、"0(正常終了)"で終了します。
本オプションは、**make**ユーティリティ等を用いている場合に、ワーニングが発生した場合にコンパイル処理を停止したいときに使用します。

-Wundefined_macro**-WUM****警告オプション**

機能 **#if**の中で未定義のマクロを使用した場合に警告します。

-Wuninitialize_variable**-WUV****警告オプション**

機能 初期化されていない **auto**変数に対してワーニングを出力します。
本オプションは、"**-Wall**"指定時にも有効になります。

補足説明: ユーザーアプリケーションにおいて、**if**文、**for**文などによる条件分岐の中で初期化される場合、コンパイラは初期化されていないと判断します。
そのため本オプションを使用した場合、警告が出力されます。

-Wunknown_pragma**-WUP****警告オプション**

- 機 能:** サポートしていない `#pragma` を使用した場合、警告を出します。
- 補足説明:** デフォルトでは、サポートされていない未知の `"#pragma"` が使用されていても警告を出しません。NC シリーズコンパイラのみを使用する場合、本オプションを使用することにより、`"#pragma"` のスペルミスなどを発見することができます。
- 注意事項:** NC シリーズコンパイラのみを使用する場合は、本オプションを常に用いてコンパイルすることを推奨します。

A.2.9 アセンブル / リンクオプション

【表A.10】にas100及びln100のオプションを選択する起動オプションを示します。

表A.10 アセンブル / リンクオプション

オプション	機能
-as100△< オプション>	アセンブルコマンドas100のオプションを指定します。 2個以上のオプションを渡す場合は、"(ダブルクォーテーション)で囲んでください。
-ln100△< オプション>	リンクコマンドln100オプションを指定します。 2個以上のオプションを渡す場合は、"(ダブルクォーテーション)で囲んでください。

-as100 "オプション"**アセンブル / リンクオプション**

機能: アセンブルコマンドas100のオプションを選択します。
2個以上のオプションを選択する場合は、"(ダブルクォーテーション)で囲んでください。

書式: nc100△-as100△ "オプション1△ オプション2"△<C言語ソースファイル名>

注意事項: as100の-, -C、-O、-PSFP、-T、およびVオプションは指定しないでください。

-ln100 "オプション"**アセンブル / リンクオプション**

機能: リンクコマンドln100のオプションを選択します。
リンクコマンドのオプションは、最大4個までオプションを選択することができます。2個以上のオプションを選択する場合は、"(ダブルクォーテーション)で囲んでください。

書式: nc100△-ln100△ "オプション1△ オプション2"△<C言語ソースファイル名>

注意事項: ln100の-, -G、-O、-ORDER、-L、-T、-Vおよび@fileオプションは指定しないでください。

A.3 起動オプションに関する注意事項

A.3.1 起動オプションの記述に関する注意事項

nc100 の起動時オプションは、アルファベットの大文字と小文字を区別します。誤って入力した場合、そのオプションによる機能は取り消されます。

A.3.2 オプションの優先順位

nc100 の起動時オプション中、

- "-c":リロケータブルファイル(拡張子.r30)を作成して処理を終える
- "-S":アセンブリ言語ソースファイル(拡張子.a30)を作成して処理を終える

を同時に指定した場合、-S オプションが優先されます。

したがって、このときはアセンブリ言語ソースファイルのみが生成されます。

付録B 拡張機能リファレンス

NC100 は、R32C/100 シリーズを用いたシステムへの組み込みを容易にするために独自の拡張機能を追加しています。付録 B では、言語仕様に関する機能以外の拡張機能の使用方法を説明します。

表B.1 拡張機能 (1/2)

拡張機能	機能の内容
near/far修飾子	<p>データをアクセスするアドレッシングモードを指定します。</p> <p>near.....64Kバイト以内の領域(00000000H~00007FFFH, 及び0FFFF8000H~OFFFFFFFFFH)のアクセス</p> <p>far.....4Gバイト以内の領域(00000000H~007FFFFFFFH, 及び0FF800000H~OFFFFFFFFFH)のアクセス</p> <ul style="list-style-type: none"> 関数は全てfar属性となります。
asm関数	<p>(1) C言語プログラム中にアセンブリ言語を直接記述できます。関数外でも記述することができます。</p> <p>記述例:</p> <pre>asm(" MOV.W #0, R0");</pre> <p>(2) 変数名を指定することができます(関数内のみ記述可能)。</p> <p>記述例1:</p> <pre>asm(" MOV.W R0, \$\$[FB]",f);</pre> <p>記述例2:</p> <pre>asm(" MOV.W R0, \$\$",s);</pre> <p>記述例3:</p> <pre>asm(" MOV.W R0, \$@",f);</pre> <p>(3) 最適化を部分的に抑止する方法の一つとしてダミーのasm関数が記述できます(関数内のみ記述可能)。</p> <p>記述例:</p> <pre>asm();</pre>
日本語文字のサポート	<p>(1) 文字列中に漢字文字を使用することができます。</p> <p>記述例:</p> <pre>L"漢字"</pre> <p>(2) 漢字文字の文字定数を使用することができます。</p> <p>記述例:</p> <pre>L'漢'</pre> <p>(3) コメント中に漢字文字を記述することができます。</p> <p>記述例:</p> <pre>/* 漢字 */</pre> <ul style="list-style-type: none"> シフトJISコード及びEUCコードをサポートしています。

表B.2 拡張機能 (2/2)

拡張機能	機能の内容
関数のデフォルト引数宣言	<p>関数の引数にデフォルト値を定義できます。</p> <p>記述例1: <pre>extern int func(int=1, char=0);</pre> </p> <p>記述例2: <pre>extern int func(int=a, char=0);</pre> </p> <ul style="list-style-type: none"> ● デフォルト値として変数を記述する時は、関数を宣言するよりも前にデフォルト値として使用する変数の宣言を行ってください。 ● デフォルト値は引数の後ろから順に埋めてください。
inline記憶クラスのサポート	<p>inline記憶クラス指定子により関数をインライン展開することができます。</p> <p>記述例: <pre>inline func(int i);</pre> </p> <ul style="list-style-type: none"> ● インライン関数を使用する前に必ずインライン関数の実体定義を行ってください。
C++風コメント	<p>C++言語風でのコメント"/"を記述できます。</p> <p>記述例: <pre>// 以降はコメントです。</pre> </p>
#pragma 拡張機能	<p>C言語から、R32C/100シリーズのハードウェア仕様を効率良く活かすための拡張機能を使用できます。</p>
アセンブラマクロ関数	<p>アセンブラ命令の一部をC言語の関数として記述することができます。</p> <p>記述例1: <pre>signed char abs_b(signed char val);</pre> </p> <p>記述例2: <pre>long int abs_(long int val);</pre> </p>

B.1 near/far修飾子

R32C/100 シリーズは、00007FFFH 番地および0FFFF8000H 番地を境界としてデータの参照 / 配置等に使用されるアドレッシングモードが変わります。NC100 は、near/far 修飾子によりアドレッシングモードの切り換えを制御できます。

B.1.1 near/far修飾子の概要

near/far 修飾子は、変数又は関数に対して使用するアドレッシングモードを選択します。

- near修飾子..... 00000000H～00007FFFHおよび0FFFF8000H～0FFFFFFFHHの領域
- far修飾子..... 00000000H～007FFFFFHおよび0FF800000H～0FFFFFFFHHの領域

near/far 修飾子は、変数又は関数の宣言時に型指定子に付加して記述します。変数及び関数の宣言時に near/far 修飾子を指定しない場合、NC100 は属性を以下のように解釈します。

- 変数の配置 near属性
- const修飾された定数の配置far属性
- 関数の配置..... far属性

また、本コンパイラはコンパイルドライバの起動オプションにより、このデフォルトの属性を変更することができます。

B.1.2 変数の宣言書式

near/far修飾子は、文法的にconst、volatile型修飾子と同様の書式で宣言時に記述します。【図B.1】に宣言時の書式を示します。

型指定子△near 又は far△変数;

図B.1 near/far修飾子を付加した変数の宣言書式

変数の宣言例を【図B.2】に、その変数のメモリ配置図を【図B.3】に示します。

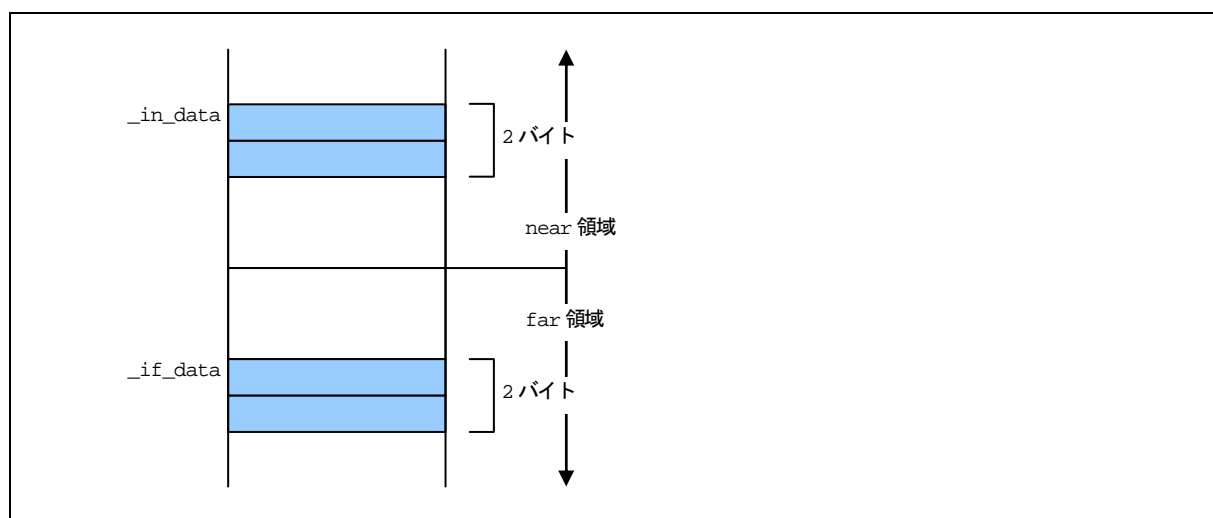
```

short near      in_data;
short far       if_data;

void   func(void)
{
    (以下省略)
    :

```

図B.2 変数の宣言例



図B.3 変数のメモリ配置

B.1.3 ポインタ型変数の宣言書式

ポインタ型変数は常に far 型(4 バイト)の変数です。ポインタ型変数の宣言に near 型を指定すると、警告メッセージ (“near pointer not supported, near qualifier ignored”) を出力し、near 修飾子を無視します。

ポインタ型の変数の宣言例を【図B.4】に示します。

例:

```
short * ptr;
```

図B.4 ポインタ型変数の宣言例 (1)

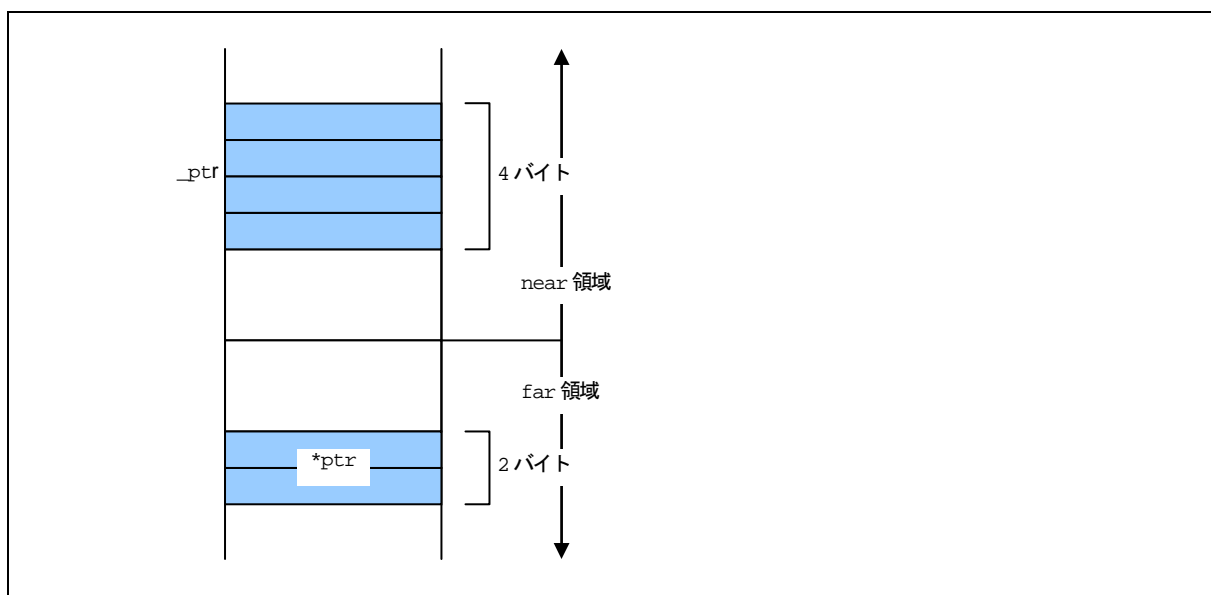
変数の配置は near、ポインタ変数の型はfar型となるため、【図B.4】の記述は【図B.5】のように解釈されます。

例:

```
short far * near ptr;
```

図B.5 ポインタ型変数の宣言例 (2)

変数ptr は、far領域にあるshort型変数を指し示す4バイトの変数です。ptr自身はnear領域に配置されます。上記例のメモリ配置を【図B.6】に示します。



図B.6 ポインタ型変数のメモリ配置

明示的にnear/farを指定した場合は、右側に記述した変数/関数を格納するアドレスのサイズを決定します。アドレスを扱うポインタ型の変数の宣言を【図B.7】に示します。

例 1:

```
short far * ptr1;
```

例 2:

```
short * far ptr2;
```

図B.7 アドレスを扱うポインタ型変数の宣言例 (1)

先にも説明したようにnear/farの指定がない場合は、変数の配置を"near"、変数の型を"far"として扱います。したがって、例 1、例 2 はそれぞれ、【図B.8】のように解釈されます。

例 1:

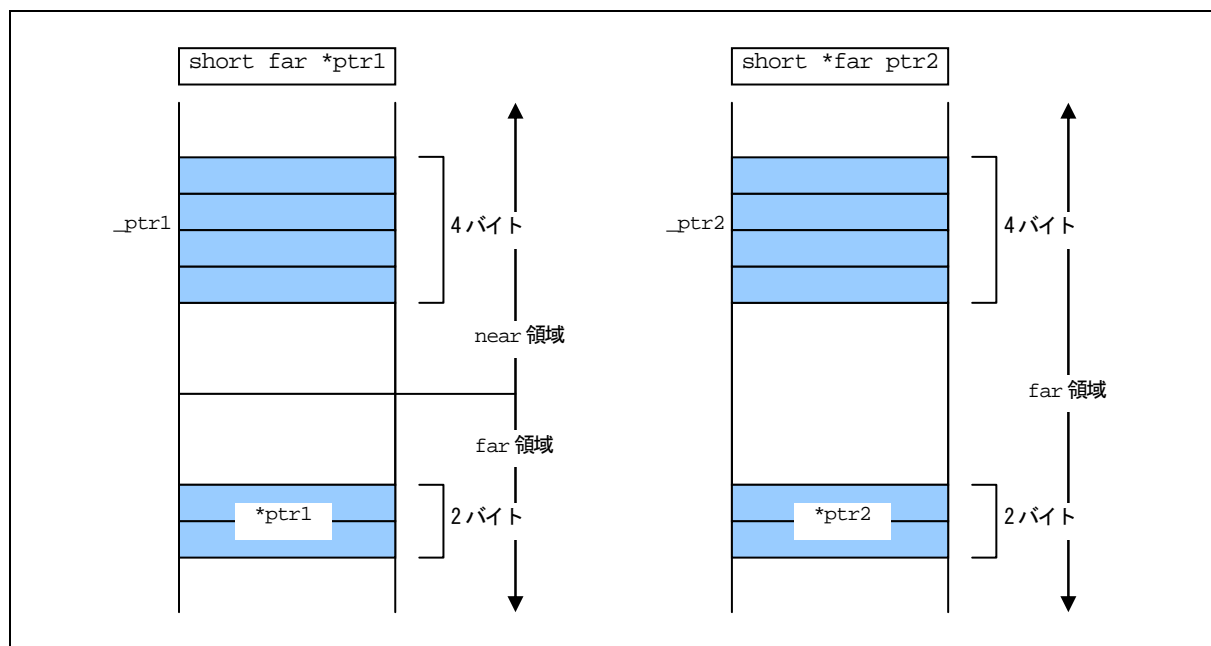
```
short far * near ptr1;
```

例 2:

```
short far * far ptr2;
```

図B.8 アドレスを扱うポインタ型変数の宣言例 (2)

例1 では変数 ptr1 は far 領域にある short 型変数を指し示す 4 バイト型の変数で、変数自身は near 領域に配置されます。
 例2 では変数 ptr2 は far 領域にある short 型変数を指し示す 4 バイト型の変数で、変数自身は far 領域に配置されます。
 例1、例2 のメモリ配置を【図B.9】に示します。



図B.9 アドレスを扱うポインタ型変数のメモリ配置

B.1.4 関数の宣言

関数の near/far 配置属性は、常に far です。関数の宣言に near 属性を指定するとワーニングメッセージ(function must be far)を出力し、near の宣言を無視します。

B.1.5 nc100 の起動オプションによるnear/farの制御

near/far属性を指定しない場合、NC100 では関数はfar属性、変数(データ)はnear属性として扱われます。NC100 の起動オプションには、変数(データ)のデフォルトを変更するオプションを用意しています(【表B.3】)。

表B.3 nc100起動オプション

起動オプション	機能
-fnear_ROM(-fNROM)	ROM データのデフォルト属性を near にします。
-ffar_RAM(-fFRAM)	RAM データのデフォルト属性を far にします。

B.1.6 nearからfarへの型変換機能

【図B.10】に示すプログラムの記述において、nearからfarへの型変換が行われます。

```
int    func( int far * );
int    far *f_ptr;
int    near n_var;

void   main(void)
{
    f_ptr = &n_var;          /* near アドレスを far ポインタに代入 */
    :
    (省略)
    :
    func ( &n_var );       /* 引数に far ポインタを持つ関数としてプロトタイプ宣言した */
                             /* 関数の呼び出し時に near アドレスの引数を指定 */
}
```

図B.10 nearからfarへの型変換

far に型変換される際、near アドレス(16 ビット)の最上位を符号として符号拡張します。

B.1.7 関数の宣言

関数は必ず far 領域に配置されます。従って、関数にはnear の宣言を行なわないでください。

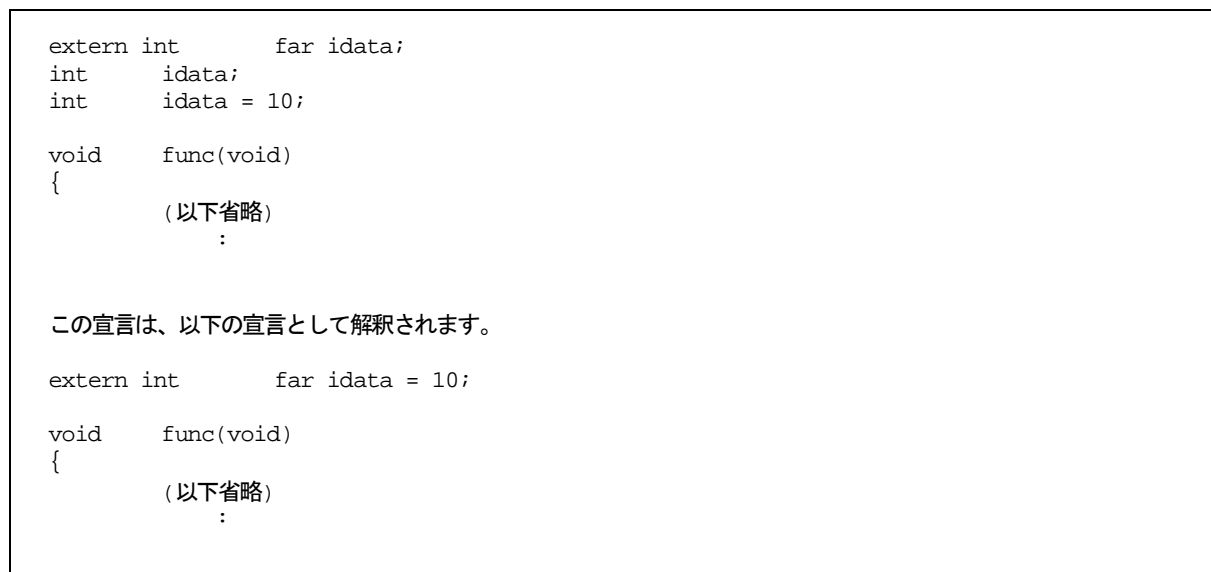
関数に対してnear属性の宣言を行なった場合は、NC100 はワーニングを出力した後、関数の属性をfarとして処理を続けます。関数に対してnear宣言を行なったときの表示例を【図B.11】に示します。

```
%nc100 -S smp.c
R32C/100 Series C Compiler V.X.XX Release XX
Copyright(C) XXXX(XXXX-XXXX). Renesas Electronics Corp.
and Renesas Solutions Corp., All rights reserved.
smp.c
[Warning(ccom):smp.c,line 3] function must be far
==> {
func
%
```

図B.11 メッセージの表示例

B.1.8 複数の宣言でnear/farの確定を行う機能

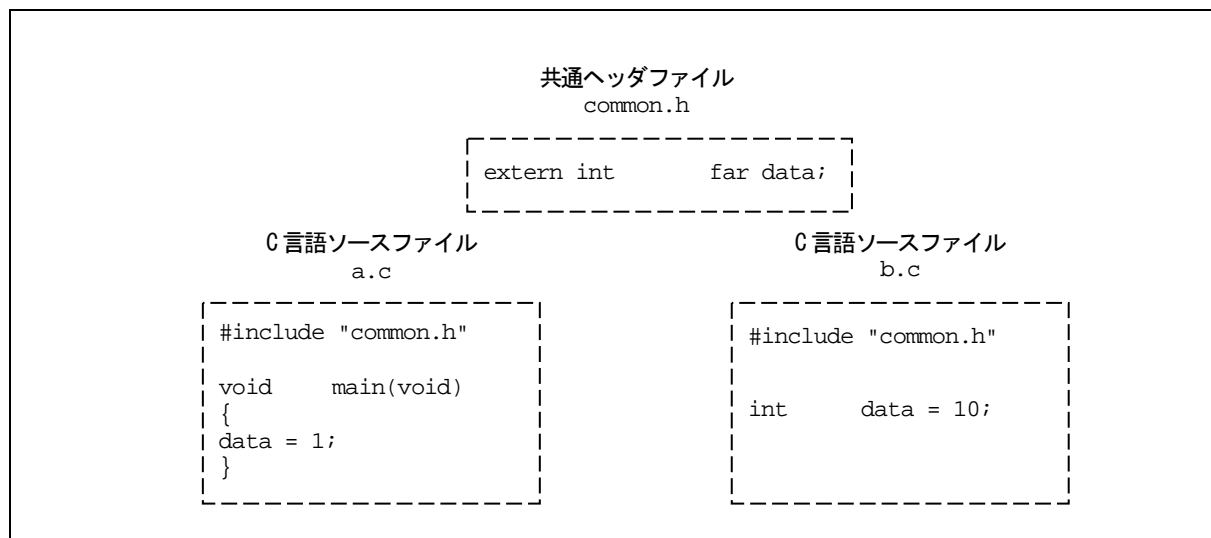
【図B.12】に示すように同一の変数に対して複数の宣言を行った場合、変数の型の情報が結合された型として解釈されます。



図B.12 変数の宣言の結合機能

この例に示すように、複数の宣言がある場合、near/farの指定はその内の1箇所で行うことで確定することができます。ただし、複数の宣言中、near と far が競合した場合はエラーとなります。

共通のヘッダファイルでnear/farの宣言を行うことにより、ソースファイル間のnear/farの整合をとることができます。



図B.13 共通ヘッダファイルの宣言例

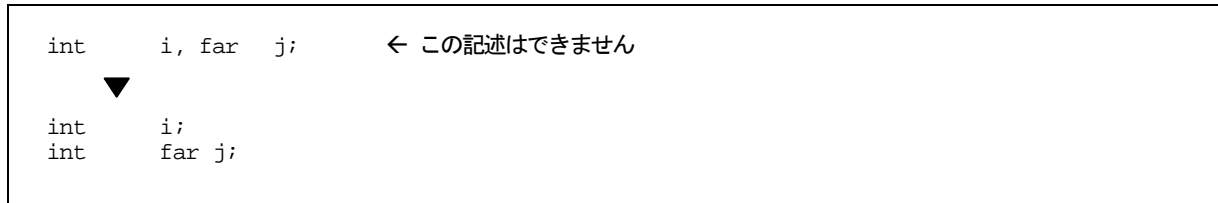
B.1.9 near/far属性に関する注意事項

(a) 関数の near/far 属性に関する注意事項

関数は必ず far 属性になります。関数は near で宣言しないでください。関数に対して near 属性の宣言を行なった場合 NC100 は警告を出力します。

(b) near/far 修飾子の文法上の注意事項

near/far修飾子は、文法上const修飾子と全く同じに扱われます。したがって、【図B.14】に示す記述はエラーとなります。



図B.14 変数の宣言例

B.2 asm関数

本コンパイラでは、C言語ソースプログラム中にアセンブリ言語のルーチン(asm関数)¹を記述することができます。asm関数では拡張機能として、C言語で記述した変数の参照機能があります。

B.2.1 asm関数の概要

asm関数は、C言語ソースプログラム中にアセンブリ言語の記述を行うときに使用します。asm関数の書式は、【図B.15】に示すようにasm(" ");のダブルクォーテーションの中にAS100の言語仕様に準じたアセンブリ言語の命令を記述します。

```
#pragma ADDRESS ta0_int 6CH
char ta0_int;

void func(void)
{
    :
    (省略)
    :
    ta0_int = 0x07;           ← タイマ A0 割り込みを許可
    asm("    FSET I");      ← 割り込み許可フラグのセット
}
```

図B.15 asm関数の記述例 (1)

また、【図B.16】に示す記述によりステートメントの前後関係を用いたコンパイラの一部の最適化処理を部分的に抑止することができます。

```
asm( );
```

図B.16 asm関数の記述例 (2)

本コンパイラで扱う asm 関数は、アセンブリ言語の記述を行うほかに、以下の拡張機能があります。

- C言語プログラムの記憶クラスauto変数のFBオフセット値をC言語の変数名で指定できます。
- C言語プログラムの記憶クラスregister変数のレジスタ名をC言語の変数名で指定できます。
- C言語プログラムの記憶クラスextern及びstatic変数のシンボル名をC言語の変数名で指定できます。

asm 関数を使用する場合の注意事項として以下の事項があります。

- asm関数内では、レジスタの内容を破壊しないでください。
- コンパイラは、asm関数内のチェックを行っていません。
- レジスタを破壊する場合は、asm関数を使用して、push命令、pop命令を記述し、退避/復帰を行ってください。

¹ 本ユーザーズマニュアルでは、表現上アセンブリ言語で記述したサブルーチンをアセンブラ関数と表記します。C言語プログラム中にasm()で記述するものはasm関数、もしくはインラインアセンブル記述と表記します。

B.2.2 auto変数のFBオフセット値の指定

C 言語で記述した記憶クラス `auto` および `register` 変数(引数を含む)は、フレームベースレジスタ(FB)に対するオフセット値で参照/配置されます (最適化等によってレジスタに割り当てられることもあります)。

【図B.17】に示す書式で記述することにより、asm関数中でスタック上に割り当てられたauto変数を使用することができます。

```
asm( " オペコード          R1 , $$ [ FB ] " , 変数名 );
```

図B.17 FBオフセット指定の記述書式

この記述形式で指定できる変数名は2つです。変数名として以下の形式をサポートしています。

- 変数名
- 配列名[整数]
- 構造体名.メンバ名(ビットフィールドメンバは除きます)

```
void func(void)
{
    short  idata;
    short  a[3];
    struct TAG{
        short  i;
        short  k;
    } s;
    :
    asm("  MOV.W   R0, $$[FB]", idata);
    :
    asm("  MOV.W   R0, $$[FB]", a[2]);
    :
    asm("  MOV.W   R0, $$[FB]", s.i);
    (以下省略)
    :
    asm("  MOV.W   $$[FB], $$[FB]", s.i, a[2]);
}
```

図B.18 FBオフセット指定の記述例

auto変数の参照例とコンパイル結果を【図B.19】に示します。

C 言語ソースファイル:

```
void func(void)
{
    short idata = 1;

    asm("    MOV.W    $$[FB], R0", idata);
    asm("    CMP.W    #00001H ,R0");
    (以下省略)
    :
}

```

アセンブリ言語ソースファイル(コンパイル結果):

```
;### # FUNCTION func
;### # FRAME AUTO ( idata) size 2, offset -4
;### # FRAME AUTO ( __PAD2) size 1, offset -1
;### # FRAME AUTO ( __PAD1) size 1, offset -2
;### # ARG Size(4) Auto Size(4) Context Size(8)
:
(省略)
;### # C_SRC : asm(" MOV.W $$[FB], R0", idata);
;#### ASM START
MOV.W -4[FB], R0
._line 5
;### # C_SRC : asm(" CMP.W #00001H ,R0");
CMP.W #00001H ,R0
;#### ASM END
(以下省略)
:

```

図B.19 auto変数の参照例

また、【図B.20】に示す書式で記述することにより、asm関数中でauto変数の1ビットのビットフィールドを使用することができます(2ビット以上のビットフィールドの操作はできません)。

```
asm( " オペコード $b[ FB ]" , ビットフィールド名 );
```

図B.20 FBオフセットビット位置指定の記述書式

この記述形式で指定できる変数名は1つです。【図B.21】に記述例を示します。

```

void    func(void)
{
    struct TAG{
        char    bit0:1;
        char    bit1:1;
        char    bit2:1;
        char    bit3:1;
    } s;

    asm("    bset    $b[FB]",s.bit1);
}

```

図B.21 FBオフセットビット位置指定の記述例

auto領域のビットフィールドの参照例とコンパイル結果を【図B.22】に示します。

```

C 言語ソースファイル

void    func(void)
{
    struct TAG{
        char    bit0:1;
        char    bit1:1;
        char    bit2:1;
        char    bit3:1;
    } s;
    asm("    bset    $b[FB]",s.bit1);
}

アセンブリ言語ソースファイル(コンパイル結果):
### #    FUNCTION func
### #    FRAME    AUTO    (    __PAD3)        size 1, offset -1
### #    FRAME    AUTO    (    __PAD2)        size 1, offset -2
### #    FRAME    AUTO    (    __PAD1)        size 1, offset -3
### #    FRAME    AUTO    (        s)            size 1, offset -4
### #    ARG Size(4)    Auto Size(4)    Context Size(8)

    .SECTION program, CODE, ALIGN
    ._file    'bit.c'
    .align
    ._line    2
### # C_SRC :    {
    .glb    _func
_func:
    enter    #04H
    ._line    9
### # C_SRC :    asm("    bset    $b[FB]",s.bit1);
##### ASM START
    bset    1,-4[FB] ; s
##### ASM END
    ._line    10
### # C_SRC :    }
    exitd

```

図B.22 auto領域のビットフィールドの参照例

B.2.3 レジスタ変数のレジスタ名の指定

C 言語で記述した記憶クラス `auto` 及び `register` の変数(引数を含む)は、コンパイラによってレジスタに割り当てられることがあります。

【図B.23】に示す書式で記述することにより、asm関数中でレジスタに割り当てられた変数を使用することができます。

1

```
asm( " オペコード $$ " , 変数名 );
```

図B.23 レジスタ変数の記述書式

この記述形式で指定できる変数名は2つです。レジスタ変数の参照例とコンパイル結果を【図B.24】に示します。

C 言語ソースファイル:

```
void func(void)
{
    register short i=1;

    asm(" mov.w $$,R1",i);
}
```

アセンブリ言語ソースファイル(コンパイル結果):

```
### # FUNCTION func
### # ARG Size(4)      Auto Size(0)      Context Size(4)

        .SECTION program, CODE, ALIGN
        ._file 'reg.c'
        .align
        ._line 2
### # C_SRC : {
        .glob _func
_func:
        ._line 3
### # C_SRC : register short i=1;
        mov.w #0001H,R0 ; i
        ._line 4
### # C_SRC : asm(" mov.w $$,R1",i);
##### ASM START
        mov.w R0,R1          ← R0 レジスタ(変数)をR1 レジスタに転送
##### ASM END
```

図B.24 register変数の参照例

本コンパイラでは、関数内で使用するレジスタ変数を動的に管理しています。ある位置でレジスタ変数として使用したレジスタが、常に同じレジスタであるとは限りません。そのため、asm関数中に直接レジスタを記述した場合、コンパイル結果により動作が異なる可能性があります。従いまして、必ずこの機能を用いてレジスタ変数を参照してください。

¹ register修飾子により強制的にレジスタに割り当てるためには、コンパイル時にオプション"-fenable_register(-fER)"を指定してください。

B.2.4 extern変数及びstatic変数のシンボル名の指定

C 言語で記述した記憶クラス `extern` 及び `static` の変数は、シンボルとして参照されます。

【図B.25】に示す書式で記述することにより、`asm`関数中で`extern`変数及び`static`の変数を使用することができます。

```
asm( " オペコード R1, $ " , 変数名 );
```

図B.25 シンボル名指定の記述書式

この記述形式で指定できる変数名は2つです。変数名として以下の形式をサポートしています。

- 変数名
- 配列名[整数]
- 構造体名.メンバ名(ビットフィールドメンバは除きます)

```
short  idata;
short  a[3];
struct TAG{
    short  i;
    short  k;
} s;

void  func(void)
{
    :
    asm("  MOV.W   R0, $$", idata);
    :
    asm("  MOV.W   R0, $$", a[2]);
    :
    asm("  MOV.W   R0, $$", s.i);
    (以下省略)
    :
}
```

図B.26 シンボル名指定の記述例

extern変数及びstatic変数の参照例を【図B.27】に示します。

```

C 言語ソースファイル:
extern short ext_val;

void func(void)
{
    static short s_val;

    asm("    mov.w  #01H,$$",ext_val);
    asm("    mov.w  #01H,$$",s_val);
}

アセンブリ言語ソースファイル(コンパイル結果):
_func:
    .line 5
;## # C_SRC :    asm(" mov.w #01H,$$",ext_val);
;#### ASM START
    mov.w  #01H,_ext_val           ← シンボル_ext_val への転送
    .line 6
;## # C_SRC :    asm(" mov.w #01H,$$",s_val);
    mov.w  #01H,___S0_s_val       ← シンボル___S0_s_val への転送
;#### ASM END
    .line 7
;## # C_SRC :    }
    rts
E1:
    .glob  _ext_val

    .SECTION bss_NEAR,DATA,ALIGN
___S0_s_val:                ;### C's name is s_val
    .blkb  2
    .END

```

図B.27 extern変数及びstatic変数の参照例

また、【図B.28】に示す書式で記述することにより、asm関数中でextern変数及びstatic変数の1ビットのビットフィールドを使用することができます(2ビット以上のビットフィールドは記述できません)。

```
asm( " オペコード $b[FB]", ビットフィールド名 );
```

図B.28 シンボル名指定の記述書式

この記述形式で指定できる変数名は1つです。【図B.29】に記述例を示します。

```

struct TAG{
    char  bit0:1;
    char  bit1:1;
    char  bit2:1;
    char  bit3:1;
} s;

void func(void)
{
    asm("    bset  $b",s.bit1);
}

```

図B.29 シンボルのビット位置指定の記述例

【図B.29】のCソースファイルのコンパイル結果を【図B.30】に示します。

```
### #   FUNCTION func
### #   ARG Size(4)      Auto Size(0)      Context Size(4)

        .SECTION program, CODE, ALIGN
        .file   'bitfield.c'
        .align
        .line   8
### # C_SRC :   {
        .glb   _func
_func:
        .line   9
### # C_SRC :   asm(" bset $b",s.bit1);
##### ASM START
        bset   1,_s          ← 構造体sのビットフィールドbit0を参照
##### ASM END
        .line   10
### # C_SRC :   }
        rts
E1:

        .SECTION bss_NEAR, DATA, ALIGN
        .glb   _s
_s:
        .blkb  1
        .END
```

図B.30 シンボルに対するビットフィールドの参照例

B.2.5 記憶クラスに依存しない指定

C言語で記述した変数を、その変数の記憶クラス(auto変数、レジスタ変数¹、extern変数、static変数)に依存することなく、asm関数中で使用することができます。

【図B.31】に示す書式で記述することにより、C言語で記述した変数をasm関数中で使用できます。²

```
asm(" オペコード R0,$@", 変数名 );
```

図B.31 変数の記憶クラスに依存しない記述書式

この記述形式で指定できる変数名は2つです。参照例とコンパイル結果を【図B.32】に示します。

C言語ソースファイル:

```
extern int      e_val;

void func(void)
{
    int      f_val;
    register int  r_val;
    static int  s_val;

    asm("  mov.w  #1, $@", e_val);      ← extern 変数の参照
    asm("  mov.w  #2, $@", f_val);      ← auto 変数の参照
    asm("  mov.w  #3, $@", r_val);      ← レジスタ変数の参照
    asm("  mov.w  #4, $@", s_val);      ← static 変数の参照
    asm("  mov.w  $@, $@", f_val,r_val);
}
```

アセンブリ言語ソースファイル(コンパイル結果):

```
.glob  _func
_func:
    enter  #04H
    .line  7
;## # C_SRC :  asm(" mov.w #1, $@", e_val);
;#### ASM START
    mov.w  #1, _e_val:16                ← extern 変数の参照
    .line  8
;## # C_SRC :  asm(" mov.w #2, $@", f_val);
    mov.w  #2, -4[FB]                  ← auto 変数の参照
    .line  9
;## # C_SRC :  asm(" mov.w #3, $@", r_val);
    mov.w  #3, R2R0                   ← register 変数の参照
    .line 10
;## # C_SRC :  asm(" mov.w #4, $@", s_val);
    mov.w  #4, __S0_s_val:16          ← static 変数の参照
    .line 11
;## # C_SRC :  asm(" mov.w $@, $@", f_val,r_val);
    mov.w  -4[FB], R2R0
;#### ASM END
```

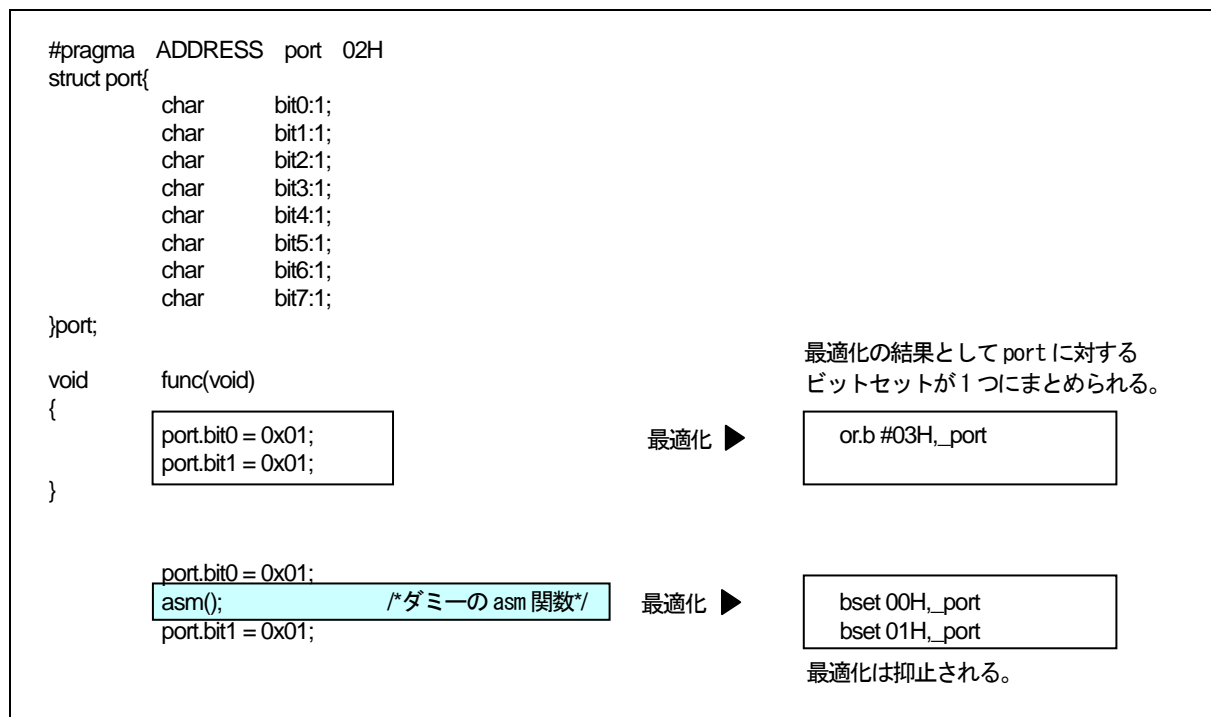
図B.32 各記憶クラスの変数の参照例

¹ register修飾子を指定しても、レジスタに割り当てられるとは限りません。

² どの記憶クラスに配置されるかは、実際にコンパイルして確認してください。

B.2.6 最適化の部分的な抑止方法

【図B.33】に示すasm関数の記述においてダミーのasm関数を記述することにより、部分的に一部の最適化を抑止することができます。



図B.33 ダミーのasm関数による最適化の抑止例

B.2.7 asm関数に関する注意事項

(a) asm関数の拡張機能

asm関数を用いて以下の処理を行うときは、必ず記述例に示す書式で記述してください。

(1) 記憶クラスが auto の変数、引数もしくは、1ビットのビットフィールドの場合

記憶クラスがautoの変数、引数もしくは、1ビットのビットフィールドをフレームベースレジスタ(FB)からのオフセット値を用いて指定しないでください。autoの変数、もしくは引数を指定する場合は、【図B.34】に示す書式で記述してください。

```

asm(" MOV.W    #01H, $$[FB]", i); ← 記憶クラス auto の変数を参照する書式です。
asm("  BSET    $$[FB]", s.bit0); ← 記憶クラス auto のビットフィールドを参照する書式です。

```

図B.34 asm関数の記述例 (1)

(2) register 記憶クラスの指定

本コンパイラでは、register記憶クラスを指定することができます。register記憶クラスで指定した変数でかつオプション-fenable_register(-fER)を指定してコンパイルした場合にasm関数でregister変数を記述する時は、【図B.35】に示す書式で記述してください。

```
asm( "    MOV.W    #0,$$", i);    ← レジスタ変数を参照する書式です。
```

図B.35 asm関数の記述例 (2)

また、オプション-O[1~5]、-OR、-OS、-OR_MAX、-OS_MAX を指定すると、コード効率の向上の為にregister渡しとなる引数をauto領域に転送を行わずにregister変数として扱う場合があります。

この場合に、asm関数で引数を指定すると変数のFBオフセット値でなくレジスタ名でアセンブリ言語を出力するので、ご注意ください。

(3) asm 関数で引数を参照する場合

本コンパイラでは、変数(引数およびauto変数を含む)の生存区間についてプログラムフローを解析して処理を行っているため、【図B.36】のようにasm関数の中で直接、引数およびauto変数を参照すると、その生存区間の管理が崩れて正しいコード出力することができません。

従って、asm関数の記述で、引数およびauto変数を参照する場合は、必ずasm関数の「\$\$,\$b,\$@」機能を使用して参照してください。

```
void    func( void )
{
    short    i, j;
    asm ( "    mov.w    -2[FB],-4[FB]");    /* j = i; */
}
```

図B.36 正しく参照することができない例

上記の場合コンパイラは、関数func内で、「i」、「j」は使用されていないと判断するため、auto変数を参照するためのフレームを構築するコードを出力しません。このため、auto変数を正しく参照できなくなります。

(4) asm 関数内でのブランチについて

本コンパイラでは、レジスタの生存区間、変数の生存区間についてプログラムフローを解析して処理を行っているため、asm関数でフローに影響を与えるようなブランチ(条件ブランチ含む)を記述しないようにしてください。

(b) レジスタについて

- asm関数内でレジスタを破壊しないでください。破壊する場合は、push命令/pop命令により、レジスタの退避/復帰を行ってください。
- SBレジスタをスタートアッププログラムで初期化後固定で使用することを前提としています。万一変更する場合は、連続するasm関数の最後で【図B.37】に示すSBレジスタを元に戻す記述を行なってください。また、変更している間に呼び出す関数や、その間に発生する割り込み処理について十分考慮してください。

```

asm( "    .SB    0);
asm( "    LDC    #0H, SB");           ← SB 変更
asm( "    MOV.W  R0, _port[SB]");
    :
    (省略)
    :
asm( "    .SB    __SB__);
asm( "    LDC    #__SB__, SB");       ← SB を元に戻す

```

図B.37 変更したスタティックベースレジスタの復帰方法

- フレームベースレジスタ(FB)は、スタックフレームポインタとして使用しますので、asm関数で変更しないでください。

(c) ラベルの記述に関する注意事項

本コンパイラが生成するアセンブルソースファイルでは、【図B.38】に示す形式の内部ラベルが出力されます。したがって、asm関数を用いてこの規定と重複する可能性のあるラベルを記述しないでください。

アルファベットの大文字 1 文字と数字で構成されるラベル名:

```

A1:
C9830:

```

_ (アンダースコア) で始まる 2 文字以上のラベル名

```

__LABEL:
__START:

```

図B.38 asm関数で記述できないラベル名の形式

B.3 日本語文字サポート

本コンパイラでは、C 言語ソースプログラム中に日本語の文字を記述することができます。

B.3.1 日本語文字の概要

日本語の文字は、アルファベット等の1バイトで表現される文字と異なり、2バイトで構成されます。NC100では、この2バイトで構成される文字を文字列、文字定数、コメントに記述することができます。記述できる文字の種類を以下に示します。

- 漢字
- ひら仮名
- 全角のカタ仮名
- 半角のカタ仮名

日本語文字の記述は、以下の漢字コード系のみで使用できます。

- EUC(ただし、1文字が3バイトで構成される外字コードは使用できません)
- シフトJIS

B.3.2 日本語文字を記述するための設定

漢字コードを使用する場合は、以下に示す環境変数を設定してください。なお、デフォルトでは、NCKIN、NCKOUT共にSJISとなっています。

- 入力コード系指定環境変数..... NCKIN
- 出力コード系指定環境変数..... NCKOUT

環境変数の設定例を【図B.39】に示します。

```
autoexec.bat の中に以下の内容を記述します。  
  
set NCKIN=SJIS  
set NCKOUT=SJIS
```

図B.39 環境変数NCKINとNCKOUTの設定例

本コンパイラでは入力漢字コードをプリプロセッサで処理します。プリプロセッサでEUCコードに変換し、その後コンパイラ内の字句解析部終段で、環境変数を基に変換し出力します。

B.3.3 文字列中の日本語文字

文字列に日本語文字を記述するときの書式を【図B.40】に示します。

```
L"漢字文字列"
```

図B.40 文字列中の漢字コード記述書式

日本語を通常の文字列と同様に"漢字文字列"と記述した場合、文字列の操作では `char` 型へのポインタ型として扱われます。したがって、2 バイト文字として操作することはできません。

2 バイト文字として扱う場合は、文字列の先頭に"L"を付加して `wchar_t` 型へのポインタ型として使用します。`wchar_t` 型は、標準ヘッダファイル `stdlib.h` の中で `unsigned short` 型に `typedef` しています。

【図B.41】に日本語の文字列の記述例を示します。

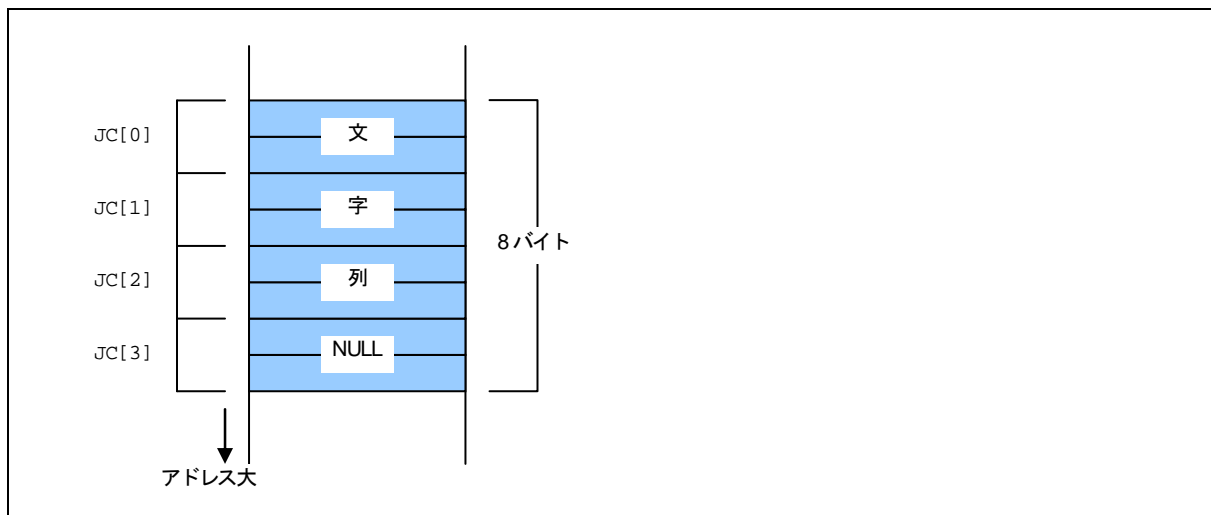
```
#include <stdlib.h>

void func(void)
{
    wchar_t JC[4] = L"文字列";

    (以下省略)
    :
```

図B.41 日本語文字列の記述例

【図B.41】中の(1)の初期化された文字列のメモリ配置を【図B.42】に示します。



図B.42 `wchar_t`型文字列のメモリ配置

B.3.4 文字定数としての日本語文字

文字定数として日本語文字を記述するときの書式を【図B.43】に示します。

```
L'漢'
```

図B.43 文字列中の漢字コード記述書式

文字列と同様に、文字定数の前に"L"を付加した場合は、`wchar_t`型として扱われます。文字定数として'文字'のように複数の文字を記述した場合は、最初の文字「文」のみが文字定数として扱われます。

【図B.44】に日本語の文字定数の記述例を示します。

```
#include <stdlib.h>

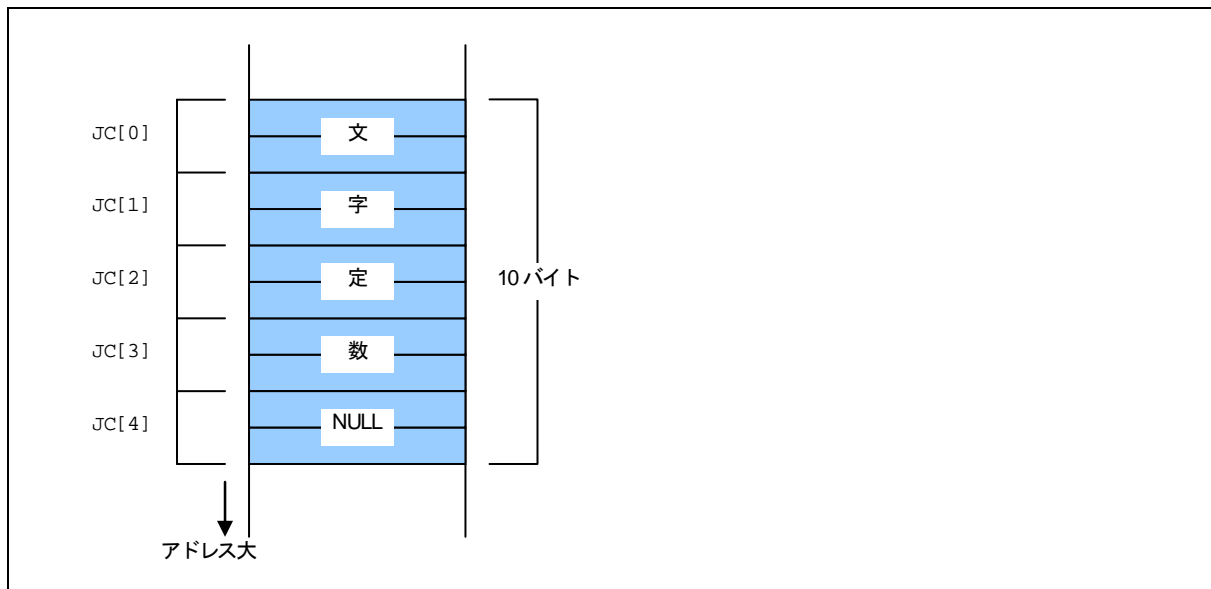
void func(void)
{
    wchar_t JC[5];

    JC[0] = L'文';
    JC[1] = L'字';
    JC[2] = L'定';
    JC[3] = L'数';

    (以下省略)
    :
```

図B.44 日本語文字定数の記述例

【図B.44】中の文字定数を代入した配列のメモリ配置を【図B.45】に示します。



図B.45 配列に代入したwchar_t型文字定数のメモリ配置

B.4 関数のデフォルト引数宣言

NC100 では、C++の機能と同様に関数の引数にデフォルト値を定義できます。この章では、関数のデフォルト引数宣言機能について説明します。

B.4.1 関数のデフォルト引数宣言の概要

NC100 では、関数のプロトタイプ宣言時に、仮引数にデフォルト値を与えることにより暗黙の実引数を使用することができます。この機能を使用することにより、関数呼び出し時に頻繁に使用する値を記述する手間を省くことができます。

B.4.2 関数のデフォルト引数宣言の書式

【図B.46】に関数のデフォルト引数宣言時の書式を示します。

```
記憶クラス指定子△型宣言子△宣言子([仮引数[=デフォルト値あるいは変数],...]);
```

図B.46 関数のデフォルト引数宣言書式

デフォルト引数宣言の例を【図B.47】に、【図B.47】で示すサンプルプログラムのコンパイル結果を【図B.48】に示します。

```
short   func( short i=1 , short j=2 );    ← 関数 func に仮引数のデフォルト値を第1引数：1、
                                           第2引数：2に宣言しています。

void    main(void)
{
    func();                               ← 実引数は第1引数：1、第2引数：2になります。
    func(3);                              ← 実引数は第1引数：3、第2引数：2になります。
    func(3,5);                             ← 実引数は第1引数：3、第2引数：5になります。
}
```

図B.47 関数のデフォルト引数の宣言例 (smp1.c)

```

;## # C_SRC :      {
      .glb      _main
_main:
      .__line   5
;## # C_SRC :      func();
      mov.w    #0002H,R1          ← 第2引数:2
      mov.w    #0001H,R0          ← 第1引数:1
      jsr     $func
      .__line   6
;## # C_SRC :      func(3);
      mov.w    #0002H,R1          ← 第2引数:2
      mov.w    #0003H,R0          ← 第1引数:3
      jsr     $func
      .__line   7
;## # C_SRC :      func(3,5);
      mov.w    #0005H,R1          ← 第2引数:5
      mov.w    #0003H,R0          ← 第1引数:3
      jsr     $func
      .__line   8
;## # C_SRC :      }
      rts
      :
      (省略)
      :

```

NC100での引数を積む順序は、関数で宣言した引数の後ろからです。
この例では引数をレジスタ渡して処理しています。

図B.48 smp1.cのコンパイル結果 (smp1.a30)

関数の引数に変数を記述することができます。

デフォルト引数の変数指定の例を【図B.49】に、【図B.49】で示すサンプルプログラムのコンパイル結果を【図B.50】に示します。

```

short   near sym ;
short   func( short i = sym);    ← デフォルトの引数を変数で指定しています。

void    main(void)
{
      func();                    ← 変数(sym)を引数として関数呼び出しを行いません。
      :
      (省略)
      :

```

図B.49 デフォルト引数の変数指定例 (smp2.c)

```

_main:
    .line 6
    mov.w    _sym:16,R0      ← 変数(sym)を引数として関数呼び出しを行いません。
    jsr     $func
    .line 7
    rts

```

図B.50 smp2.cのコンパイル結果 (smp2.a30)

B.4.3 関数のデフォルト引数宣言の規定事項

関数のデフォルト引数の宣言を行なう場合は、以下の点に注意してください。

(a) 複数の引数にデフォルト値を指定する時

関数の引数が複数ある場合にデフォルト値を指定する場合は、必ず引数の後ろから埋めてください。【図B.51】に不適切な記述の例を示します。

```

void    func1(int i, int j=1, int k=2);      /* 正しい記述です*/
void    func2(int i, int j, int k=2);      /* 正しい記述です*/
void    func3(int i = 0, int j, int k);    /* 誤った記述です*/
void    func4(int i = 0, int j, int k = 1); /* 誤った記述です*/

```

図B.51 プロトタイプ宣言の記述例

(b) 変数のデフォルト値を指定する時

デフォルト値として変数を指定する場合は、指定する変数の宣言を行なった後に関数のプロトタイプ宣言を行なってください。関数のプロトタイプ宣言を行なった時点で、宣言していない変数を引数のデフォルト値に指定した場合は、エラーとして処理します。

B.5 inline関数宣言

C++風に `inline` 記憶クラスを指定することができます。関数に対して `inline` 記憶クラスを指定することにより関数をインライン展開することができます。

B.5.1 inline記憶クラスの概要

`inline` 記憶クラス指定子は、関数に対してインライン展開される関数であることを宣言します。`inline` 記憶クラス指定した関数は、アセンブリ言語レベルでは直接コードが埋め込まれます。

B.5.2 inline記憶クラスの宣言書式

`inline`記憶クラス指定子は、文法的に`static`、`extern`型記憶クラス指定子と同様の書式で宣言時に記述します。【図B.52】に宣言時の書式を示します。

```
inline△型指定子△関数;
```

図B.52 inline記憶クラスの宣言書式

関数の宣言例を【図B.53】に、コンパイル結果【図B.54】を示します。

```
inline short      func(short i)    ← インライン関数の宣言及び定義部
{
    return i++;
}

void main(void)
{
    short s;

    s = func(s);                  ← インライン関数呼び出し部
}
```

図B.53 インライン関数のサンプルプログラム (sample.c)

```

        .SECTION program, CODE, ALIGN
        ._file 'sample.c'
        .align
        ._line 7
;## # C_SRC : {
        .glb  _main
_main:
        enter  #04H
        ._line 10
;## # C_SRC :      s = func(s);
        mov.w  -4[FB], R0      ; s
        ._line 2
;## # C_SRC : {
        mov.w  R0, -2[FB]      ; i
        ._line 3
;## # C_SRC :      return i++;
        mov.w  R0, R1
        add.w  #0001H, R0
        ._line 10
;## # C_SRC :      s = func(s);
        mov.w  R1, -4[FB]      ; s
        ._line 11
;## # C_SRC : }
        exitd
E1:
        .END

```

← インライン関数が埋め込まれている

図B.54 サンプルプログラムのコンパイル結果 (sample.a30)

B.5.3 inline記憶クラスの規定事項

inline 記憶クラス指定場合は、以下の点に注意してください。

(1) インライン関数の引数について

インライン関数の引数に構造体や共用体を使用する事はできません。これらを使用した場合は、コンパイルエラーになります。

(2) インライン関数の間接呼び出しについて

インライン関数の間接呼び出しをすることはできません。間接呼び出しの記述を行った場合は、コンパイルエラーになります。

(3) インライン関数の再帰呼び出しについて

インライン関数の再帰呼び出しをすることはできません。再帰呼び出しの記述を行った場合、コンパイルエラーになります。

(4) インライン関数の定義について

関数に対してinline記憶クラスを指定する場合は、宣言の記述の後に必ず実体定義を行なってください。実体定義は、必ず同一ファイル内に記述してください。本コンパイラでは、【図B.55】の記述はエラーになります。

```
inline void    func(int i);
```

```
void    main( void )  
{  
    func(1);  
}
```

エラーメッセージ:

```
[Error(ccom):sample.c,line 5] inline function's body is not declared previously  
==> func(1);  
Sorry, compilation terminated because of these errors in main().
```

図B.55 インライン関数の不適切な記述例 (1)

また、ある関数を通常の間数として使用した後に、その関数をインライン関数として定義した場合は、inlineの指定は無効になり、すべてstaticな関数として扱います(【図B.56】)。

```
int    func(int i);
```

```
void    main( void )  
{  
    func(1);  
}
```

```
inline int    func(int i)  
{  
    return i;  
}
```

ワーニングメッセージ:

```
[Warning(ccom):smp.c,line 10] inline function is called as normal function before,change  
to static function.
```

図B.56 インライン関数の不適切な記述例 (2)

(5) インライン関数のアドレスについて

インライン関数は、アドレスを持ちません。このため、インライン関数に対して&演算子を使用した場合は、エラーになります(【図B.57】)。

```
inline int      func(int i)
{
    return i;
}

void main(void)
{
    int      (*f)(int);

    f = &func;
}
```

エラーメッセージ:

```
[Error(ccom):sample.c,line 10] can't get inline function's address by '&' operator
==> f = &func;
Sorry, compilation terminated because of these errors in main().
```

図B.57 インライン関数の不適切な記述例 (3)

(6) static データの宣言

インライン関数内でstaticデータを宣言した場合は、宣言したstaticデータの実体はファイル単位で確保されます。

このため、複数のファイルにまたがったインライン関数ではデータのアクセス領域が異なります。インライン関数内で使用するstaticデータは関数外で宣言してください。

本コンパイラでは、インライン関数内でstatic宣言をした場合は、警告になります。また、インライン関数内のstatic宣言は推奨しません(【図B.58】)。

```
inline int      func( int j)
{
    static int      i = 0;

    i++;
    return i + j;
}
```

ワーニングメッセージ:

```
[Warning(ccom):smp.c,line 3] static valuable in inline function
==> static int i = 0;
```

図B.58 インライン関数の不適切な記述例 (4)

(7) デバッグ情報について

本コンパイラではインライン関数に対するC言語レベルのデバッグ情報を出力しません。このため、インライン関数のデバッグはアセンブリ言語レベルで行なうことになります。

B.6 コメント "//"の概要

NC100 では、"/*"と"*/"で記述するコメント以外にC++言語風のコメント "//"を記述することができます。

B.6.1 コメント "//"の概要

C 言語でコメントは、"/*"と"*/"の間に記述する必要があります。C++言語でのコメントは、同一行で"/"以降の記述がすべてコメントになります。

B.6.2 コメント "//"の書式

行中で"/"を記述した場合同一行の以降の記述は、コメントとして扱われます。【図B.59】に書式を示します。

```
// コメント
```

図B.59 コメントの記述書式

コメントの記述例を【図B.60】に示します。

```
void    func(void)
{
    int i;        /* コメントです */
    int j;        // コメントです
    :
    (省略)
    :
}
```

図B.60 コメントの記述例

B.6.3 "//" と "/*"の優先順序

"//"と"/*"、"*/"の優先順序は、「先に出現した方」が優先されます。したがって、"/" から改行コードの間に記述された"/*" は、コメント開始の意味を持ちません。

また、"/*" ~ "*/" の間に記述された "//" も、コメント開始の意味を持ちません。

B.7 #pragma 拡張機能

B.7.1 #pragma 拡張機能の一覧

#pragma に関する拡張機能の内容と規定を一覧表で示します。

(a) メモリ配置に関する拡張機能

表B.4 メモリ配置に関する拡張機能 (1/2)

拡張機能	機能の内容
#pragma EXTMEM	0番地相対アドレッシングではアクセスできない領域にデータまたは関数を配置することを宣言します。 記述形式: #pragma EXTMEM△変数名 #pragma EXTMEM△関数名() 記述例: #pragma EXTMEM val #pragma EXTMEM func()
#pragma MONITORn	RAMモニタ用の特定セクションに配置するデータであることを宣言します。 記述形式: #pragma MONITOR1△変数名 記述例: #pragma MONITOR1 val
#pragma ROM	指定した変数をromセクションに配置します。 記述形式: #pragma ROM△変数名 記述例: #pragma ROM val
#pragma SB16DATA	SB相対16ビットディスプレースメントアドレッシングを使用するデータであることを宣言します。 記述形式: #pragma SB16DATA △変数名 記述例: #pragma SB16DATA sym_data
#pragma SBDATA	SB相対8ビットディスプレースメントアドレッシングを使用するデータであることを宣言します。 記述形式: #pragma SBDATA△変数名 記述例: #pragma SBDATA sym
#pragma SECTION	本コンパイラが生成するセクション名を変更します。 記述形式: #pragma SECTION△既定セクション名△変更セクション名 記述例: #pragma SECTION bss nonval_data

表B.5 メモリ配置に関する拡張機能 (2/2)

拡張機能	機能の内容
#pragma STRUCT	<p>(1) 指定したタグを持つ構造体のパックを禁止します。</p> <p>記述形式: #pragma STRUCT△構造体のタグ名△unpack</p> <p>記述例: #pragma STRUCT TAG1 unpack</p> <p>(2) 指定したタグを持つ構造体のメンバの並べ替えを行い、偶数サイズのメンバを先に配置します。</p> <p>記述形式: #pragma STRUCT△構造体のタグ名△arrange</p> <p>記述例: #pragma STRUCT TAG1 arrange</p>

(b) 組み込み機器に関する拡張機能

表B.6 組み込み機器に使用するための拡張機能 (1/2)

拡張機能	機能の内容
#pragma ADDRESS	<p>変数を絶対アドレスに割り付けます。</p> <p>記述形式: #pragma ADDRESS△変数名△絶対アドレス</p> <p>記述例: #pragma ADDRESS port0 2H</p>
#pragma DMAC	<p>外部変数に対して、DMACレジスタを割り付けます。</p> <p>記述形式: #pragma DMAC△変数名△DMACレジスタ名</p> <p>記述例: #pragma DMAC dsa0 DSA0</p>
#pragma INTCALL	<p>ソフトウェア割り込み(int 命令)で呼び出す関数を宣言します。</p> <p>記述形式1: #pragma INTCALL△INT番号△アセンブラ関数名 (レジスタ名)</p> <p>記述例1: #pragma INTCALL 25 func(R0, R1)</p> <p>記述形式2: #pragma INTCALL△INT番号△C言語関数名()</p> <p>記述例2: #pragma INTCALL 25 func()</p> <ul style="list-style-type: none"> ● 本宣言を行う前に、必ず関数のプロトタイプ宣言を行ってください。

表B.7 組み込み機器に使用するための拡張機能 (2/2)

拡張機能	機能の内容
#pragma INTERRUPT	<p>C言語で記述した割り込み処理関数を宣言します。この宣言により、関数の出入り口で割り込み処理関数の手続きを行うコードを生成します。</p> <p>記述形式:</p> <pre>#pragma INTERRUPT Δ[B E F R M] Δ割り込み処理関数名 #pragma INTERRUPT Δ[B E F R] Δ割り込みベクタ番号 Δ割り込み処理関数名 #pragma INTERRUPT Δ[B E F R] Δ割り込み処理関数名(vect=割り込みベクタ番号)</pre> <p>記述例:</p> <pre>#pragma INTERRUPT int_func #pragma INTERRUPT /B int_func #pragma INTERRUPT 10 int_func #pragma INTERRUPT /E 10 int_func #pragma INTERRUPT int_func (vect=10) #pragma INTERRUPT /R int_func</pre>
#pragma PARAMETER	<p>アセンブラで記述された関数を呼び出す際に、その引数をレジスタを介して渡すことを宣言します。</p> <p>記述形式:</p> <pre>#pragma PARAMETER Δ関数名(レジスタ名)</pre> <p>記述例:</p> <pre>#pragma PARAMETER asm_func(R0, R1)</pre> <ul style="list-style-type: none"> ● 本宣言を行う前に、必ず関数のプロトタイプ宣言を行ってください。

(c) その他の拡張機能

表B.8 その他の拡張機能

拡張機能	機能の内容
#pragma __ASMMACRO	<p>アセンブラのマクロで定義した関数を宣言します。</p> <p>記述形式: #pragma __ASMMACRO△関数名 (レジスタ名)</p> <p>記述例: #pragma __ASMMACRO max_w(R0,R2)</p>
#pragma ASM #pragma ENDASM	<p>アセンブリ言語で記述を行う領域を指定します。</p> <p>記述形式: #pragma△ASM #pragma△ENDASM</p> <p>記述例: #pragma ASM mov.w R0,R1 add.w R1,02H #pragma ENDASM</p>
#pragma JSRA	<p>JSR命令をJSR.A命令に固定して関数を呼び出します。</p> <p>記述形式: #pragma JSRA △関数名</p> <p>記述例: #pragma JSRA func</p>
#pragma JSRW	<p>JSR命令をJSR.W命令に固定して関数を呼び出します。</p> <p>記述形式: #pragma JSRW△関数名</p> <p>記述例: #pragma JSRW func</p>
#pragma PAGE	<p>アセンブラリスティングファイルの改ページの指定を行います。</p> <p>記述形式: #pragma△PAGE</p> <p>記述例: #pragma PAGE</p>

B.7.2 メモリ配置に関する拡張機能

本コンパイラは、以下に示すメモリの配置に関する拡張機能を持っています。

#pragma EXTMEM

0番地相対アドレッシング除外宣言機能

- 機能** 0番地相対アドレッシングではアクセスできない領域に配置する変数または関数を宣言します。
- 書式** #pragma EXTMEM△変数名
#pragma EXTMEM△関数名()
- 解説** 外部変数の直接アクセス、または関数の直接呼び出しで通常使用される0番地相対アドレッシングでは、00800000H番地から0FF7FFFFFH番地の範囲にアクセスすることができません。
#pragma EXTMEMは、変数または関数をこの領域に配置することを宣言します。
#pragma EXTMEM宣言された変数および関数は、アドレスレジスタ相対アドレッシングなどによりアクセスされます。なお、#pragma EXTMEM宣言された変数および関数も、そのアドレスを通常のfarポインタで扱うことができます。
- 規定**
- (1) 変数名および関数名以外を宣言した場合、無視されます。
 - (2) 関数内で宣言したstatic変数は、対象となりません。
 - (3) 宣言時のnearおよびfar修飾よりも本機能が優先されます。
 - (4) 本拡張機能は、単一の変数または関数について、他の#pragma拡張機能と併用できません。

使用例

```
#pragma EXTMEM extfunc()
#pragma EXTMEM extvar;

short extfunc( void );
short far extvar;
short far *p;

void func( void )
{
    extvar = extfunc();
    p = &extvar;
}
```

図B.61 #pragma EXTMEM宣言の使用例

備考 本機能により、以下のセクションが生成されます。

名称	属性	内容
data_EXT	DATA	#pragma EXTMEM宣言した初期値ありデータ
bss_EXT	DATA	#pragma EXTMEM宣言した初期値なしデータ
rom_EXT	ROMDATA	#pragma EXTMEM宣言したconst修飾データ
data_EXT_I	ROMDATA	data_EXTセクションの初期値
program_EX	CODE	#pragma EXTMEM宣言した関数のコード

#pragma MONITORn

RAMモニタ領域の配置指定機能

- 機能** 指定された外部変数をRAMモニタ領域専用のセクションに配置することを宣言します。
- 書式** #pragma MONITOR[n]△外部変数名 (nは、1～4)
- 規定**
- (1) 指定できる変数は外部変数および外部static変数のみです。
 - (2) #pragma MONITOR[n]で宣言された外部変数の領域は、下記のセクションに割り当てられます。
 - data_MON[n] (初期値を持つ外部変数を配置)
 - bss_MON[n] (初期値を持たない外部変数を配置)
 - data_MON[n]_INIT (初期値を持つ外部変数の初期値を配置)
 - (3) #pragma MONITOR[n]の宣言は外部変数の定義よりも前に行う必要があります。
 - (4) #pragma MONITOR[n]で宣言した外部変数は他の#pragma拡張機能と併用することはできません。ただし、#pragma SBDATAと#pragma MONITOR[n]が同時に指定された場合は#pragma SBDATAが優先されます。この際、ワーニングは出力されません。
- 注意**
- (1) #pragma MONITOR[n]はコンパイラが生成するオペコードに影響を与えません。変数のnear/far属性に注意してください。
 - (2) RAMモニタ領域専用のセクションにnear/far属性が異なる外部変数が混在してもエラー・警告にはなりません。変数のnear/far属性に注意してください。
 - (3) RAMモニタ領域専用のセクションにサイズ制限はありません。
 - (4) #pragma MONITOR[n]により割り当てられるセクションの配置アドレス、および外部変数の初期値設定処理はスタートアッププログラムで記述してください。
 - (5) 同一外部変数に対して#pragma MONITOR[n]を複数回指定した場合は、最初に指定された#pragma MONITOR[n]が有効になります。
 - (6) #pragma SECTIONの影響は受けません。
 - (7) #pragma MONITOR[n]のnが1～4以外の場合は無効になります。コンパイルオプション-Wunknown_pragma[-WUP]または-Wallを指定した場合はワーニングを出力します。
 - (8) ROM属性を持つ外部変数は#pragma MONITOR[n]の対象外になります。ただし、コンパイルオプション-fconst_not_ROM[-fCNR]を指定している場合は#pragma MONITOR[n]対象になります
 - #pragma MONITOR1 i
 - const int i; <==== 無効
 - (9) 本機能で変数の配置を変更しても、生成コードのアドレッシングモードは変更されません。本来near属性のRAM領域(00000000H番地から00007FFFH番地)に格納される変数の配置を変更した場合には、当該変数の含まれるセクションはnear領域に配置する必要があります。(初期値のROMDATA属性セクションを除きます。)

#pragma MONITORn

RAMモニタ領域の配置指定機能

使用例:

```
#pragma MONITOR1 i
#pragma MONITOR1 c

short i;
short j = 0x0100;
```

図B.62 #pragma MONITORn宣言の使用例

#pragma ROM

romセクションへの配置機能

機能 指定データ(変数)をromセクションに配置します。

書式 #pragma ROM△変数名

解説 この拡張機能は、以下の条件のどちらか一方を満たす変数に対する#pragma ROMのみ有効となります。

- 関数外で定義されたextern宣言されていない(実体を定義した)変数
- 関数内でstatic宣言された変数

- 規定**
- (1) 変数名以外を指定した場合、無効となります。
 - (2) #pragma ROM宣言の二重定義はエラーとなりません。
 - (3) 初期化式を記述しない場合は、初期値を0としてromセクションに配置されます。

使用例

```

C言語ソースプログラム:

#pragma ROM i
unsigned short  i;                                ← (1)の条件を満たす変数 i

void  func(void)
{
    static short  i = 20;                        ← (2)の条件を満たす変数 i
    :
    (以下省略)

アセンブリ言語ソースプログラム:

        .SECTION rom_FAR,ROMDATA,ALIGN
        .glob  _i
__i:
        .byte  00H
        .byte  00H
__S0_i: ;### C's name is i                       ← (2)の条件を満たす変数 i
        .word  0014H

```

図B.63 #pragma ROM宣言の使用例

#pragma SB16DATA

SB相対16ビットディスプレイースメントアドレッシング使用変数宣言機能

- 機能** SB相対16ビットディスプレイースメントアドレッシングを使用する変数データであることを宣言します。
- 書式** #pragma SB16DATA△変数名
- 解説** R32C/100シリーズでは、SB相対アドレッシングを使用すると効率の良い命令を選択することができます。SB相対アドレッシングでアクセスするセクションを **far**領域に配置した場合、#pragma SB16DATAを使用する事により、変数のデータ参照時に、SB相対アドレッシングを16ビットディスプレイースメントで使用することを宣言します。
この機能によりROM効率の良いコードを生成できます。
- 規定**
- (1) #pragma SB16DATAを使用する場合、SB相対アドレッシングでアクセスするセクションを **far**領域に配置する必要があります。そのためスタートアップファイルでのセクション配置の指定を変更する必要があります。本機能を使用するためのセクション配置については、"第2章 コンパイラの基本的な使い方 2.2.2 スタートアッププログラムのカスタマイズ"および、"第2章 コンパイラの基本的な使い方 2.2.3 メモリ配置のカスタマイズ"を参照してください。
 - (2) 同一変数に対して #pragma SB16DATAと #pragma SB16DATA を同時に指定できません。
 - (3) 変数名以外を指定した場合、無効となります。
 - (4) 指定した変数が関数内で宣言されたstatic変数の場合、無効となります。
 - (5) #pragma SB16DATAを宣言した変数は、領域確保の際にSB16DATA属性のセクションに配置されます。
 - (6) ROMデータに対して#pragma SB16DATAを宣言した場合、無効となります。¹

使用例

```
#pragma SB16DATA sym_data
int      far sym_data;

void     func( void )
{
    sym_data = 1;
}
```

図B.64 #pragma SB16DATA宣言の使用例

- 備考** NC100では、SBレジスタはリセット後初期化され、以降は固定として使用することを前提としています。

¹ ROMデータに対して"#pragma SB16DATA"宣言を行なわないでください。

#pragma SBDATA

SB相対8ビットディスプレイacementsメントアドレッシング使用変数宣言機能

機能 SB相対8ビットディスプレイacementsメントアドレッシングを使用する変数データであることを宣言します。

書式 #pragma SBDATA△変数名

解説 R32C/100シリーズでは、SB相対アドレッシングを使用すると効率の良い命令を選択することができます。
#pragma SBDATAでは、変数のデータ参照時にSB相対アドレッシングを使用することを宣言します。この機能によりROM効率の良いコードを生成できます。

- 規定**
- (1) #pragma SBDATAを宣言した変数は、アセンブラの疑似命令.SBSYMで宣言されます。
 - (2) 変数名以外を指定した場合、無効となります。
 - (3) 指定した変数が関数内で宣言されたstatic変数の場合、無効となります。
 - (4) #pragma SBDATAを宣言した変数は、領域確保の際にSBDATA属性のセクションに配置されます。
 - (5) 同一変数に対して #pragma SBDATAと #pragma SB16DATA を同時に指定できません。
 - (6) ROMデータに対して#pragma SBDATAを宣言した場合、SBDATA属性のセクションに配置されません。

使用例

```
#pragma SBDATA sym_data
struct sym_data{
    char    bit0:1;
    char    bit1:1;
    char    bit2:1;
    char    bit3:1;
    char    bit4:1;
    char    bit5:1;
    char    bit6:1;
    char    bit7:1;
}sym_data;

void    func( void )
{
    sym_data.bit1 = 0;
    :
    (省略)
    :
```

図B.65 #pragma SBDATA宣言の使用例

補足 NC100では、SBレジスタはリセット後初期化され、以降は固定として使用することを前提としています。

#pragma SECTION

セクション名変更機能

機能 コンパイラが生成するセクション名を変更します。

書式 #pragma SECTION△既定セクション名△変更セクション名

解説 この宣言をprogramセクションに対して行った場合は、その#pragma宣言以降に記述された関数のセクション名を変更します。また、この宣言をデータ(data、bss 及び rom)セクションに対して行った場合は、そのファイル中で実体を定義した全てのデータセクション名を変更します。なお、この機能を使用してセクション名を変更した場合、セクション名の追記 / 変更、必要があれば該当するセクション領域の初期化等、スタートアッププログラムを変更してください。

- programセクション、dataセクション、romセクションおよびbssセクションは、同一ファイル内で複数回セクション名を変更することができます。
- その他のセクションは、複数回セクション名を変更することができません。

使用例

C言語ソースプログラム:

```
#pragma SECTION program pro1 ← program セクション名を pro1 に変更
void func( void );
:
(以下省略)
```

アセンブリ言語ソースプログラム:

```
### FUNCTION func
.section pro1, CODE, ALIGN ← pro1 セクションに配置
.file 'smp.c'
.line 9
.glob _func
_func:
```

data セクションに対して複数回変更する場合の例:

```
#pragma SECTION data data1
int i; ← data1_NEAR セクションに配置

void func(void)
{
(省略)
}

#pragma SECTION data data2
int j; ← data2_NEAR セクションに配置

void sub(void)
{
(省略)
}
```

図B.66 #pragma SECTION宣言の使用例

#pragma SECTION

セクション名変更機能

備考: セクションの名称を変更するときには、セクション名の後ろにセクションの配置属性(_NEAR、_NEAR_INIT等)が付加されるのでご注意ください。

注意事項: 文字列データおよび初期値のないconstデータは、最後に宣言された"rom"セクション名で出力されます。

#pragma STRUCT

構造体配列制御機能

- 機能** (1) 構造体のパックを禁止します。
 (2) 構造体メンバ配置の並び換えを行います。
- 書式** (1) #pragma STRUCT△構造体のタグ名△unpack
 (2) #pragma STRUCT△構造体のタグ名△arrange
- 解説** 本コンパイラでは、構造体はパックされます。例えば、【図B.67】に示す構造体のメンバは、宣言された順にパディング(透き間)を入れずに配置されます。

メンバ名	データ型	データサイズ	配置位置 (オフセット)
i	short	16 ビット	0
c	char	8 ビット	2
j	short	16 ビット	3

図B.67 構造体メンバの配置例 (1)

- 規定** (1) パックの禁止
- 本コンパイラでは、拡張機能として構造体メンバの配置を制御することができます【図B.67】に示した構造体を#pragma STRUCTでパックを禁止したときのメンバの配置例を【図B.68】に示します。

メンバ名	データ型	データサイズ	配置位置 (オフセット)
i	short	16 ビット	0
c	char	8 ビット	2
j	short	16 ビット	3
パディング	(char)	8 ビット	-

図B.68 構造体メンバの配置例 (2)

【図B.68】に示したように、構造体メンバのサイズの合計が奇数バイトのとき、#pragma STRUCTを用いることにより最後のメンバ配置位置の後に、1バイトのパディングが入ります。したがって、#pragma STRUCTでパックを禁止したときの構造体は、すべて偶数バイトのサイズとなります。

#pragma STRUCT

構造体配列制御機能

規 定 (2) メンバ配置の並び換え

本コンパイラでは、拡張機能として構造体の偶数サイズのメンバを先に配置し、奇数サイズのメンバを後に配置することができます。【図B.68】に示した構造体を#pragmaSTRUCTで配置を並び替えたときの配置例を【図B.69】に示します。

<pre>struct s { short i; char c; short j; };</pre>	メンバ名	データ型	データサイズ	配置位置 (オフセット)
	i	short	16 ビット	0
	j	short	16 ビット	2
	c	char	8 ビット	4

図B.69 構造体メンバの配置例 (3)

パックの禁止及びメンバ配置の並び替えのための#pragma STRUCTは、必ず構造体のメンバ定義を行う前に宣言してください。

使用例

```
#pragma STRUCT TAG unpack
struct TAG {
    int    i;
    char   c;
} s1;
```

図B.70 #pragma STRUCT宣言の使用例

B.7.3 組み込み機器に関する拡張機能の使用方法

本コンパイラは、以下に示す組み込みに関する拡張機能を持っています。

#pragma ADDRESS

入出力変数の絶対アドレス割り付け機能

- | | |
|------|--|
| 機 能: | 変数を絶対アドレスに割り付けます。 |
| 書 式: | #pragma ADDRESS△変数名△絶対アドレス |
| 解 説: | <p>この宣言により指定された絶対アドレスは、文字列としてアセンブラファイルに展開され、疑似命令.EQUを用いて定義されます。したがって、数値の記述形式はアセンブラに依存します。アセンブラの数値表現を以下に示します。</p> <ul style="list-style-type: none"> ● 2進数数値の最後に'B'又は'b'を付けます。 ● 8進数数値の最後に'O'又は'o'を付けます。 ● 10進数整数のみで記述します。 ● 16進数数値の最後に'H'又は'h'を付けます。先頭が英文字(A~F)で始まる場合は先頭に0を付けます。 |
| 規 定: | <ol style="list-style-type: none"> (1) #pragma ADDRESSにより指定された変数に対するextern、static等の記憶クラスはすべて無効になります。 (2) #pragma ADDRESSにより指定された変数は、関数外で定義された変数に対してのみ有効になります。 (3) #pragma ADDRESS宣言を行う前に宣言した変数に対しても有効になります。 (4) 変数名以外を指定した場合、無効となります。 (5) #pragma ADDRESS宣言の二重定義はエラーとなりませんが、後に宣言したアドレスが有効になります。 (6) 初期化式を記述した場合はワーニングとなり、記述した初期化式は無効となります。 (7) #pragma ADDRESS又は#pragma EQUは通常、I/O変数に対して使用するため、volatile指定が無くても、volatile指定がされているものとして処理されます。 |

使用例:

```
#pragma ADDRESS port 24H
int    io;

void    func(void)
{
    io = 10;
}
```

図B.71 #pragma ADDRESS宣言

#pragma ADDRESS

入出力変数の絶対アドレス割り付け機能

注意事項: 【図B.72】のように、変数が#pragma ADDRESS の指定より先に使用されている場合は、#pragma ADDRESS の指定は無効です。

```
char    port;

void    func(void)
{
    port = 0;          /* #pragma ADDRESS の指定の前に変数を使用 */
}

#pragma ADDRESS port 100H
```

図B.72 #pragma ADDRESSの指定が無効になる場合

備考: この書式の絶対アドレスはC言語の数値表現で記述します。

#pragma DMAC

外部変数のDMACレジスタ割り付け機能

機能 指定した外部変数に対して、CPU内部のDMACレジスタを割り付けます。

書式 #pragma DMAC△変数名△DMACレジスタ名

- 規定** (1) #pragma DMAC の記述前に、指定する変数を宣言しておく必要があります。
 (2) 指定できる DMAC レジスタ、および変数の型は以下の通りです。

レジスタ名	DMD0	DCT0	DCR0	DDA0	DDR0	DSA0	DSR0
	DMD1	DCT1	DCR1	DDA1	DDR1	DSA1	DSR1
	DMD2	DCT2	DCR2	DDA2	DDR2	DSA2	DSR2
	DMD3	DCT3	DCR3	DDA3	DDR3	DSA3	DSR3
使用できる型	unsigned long			任意の型への far ポインタ ただし、関数へのポインタは不可			

- (3) 同一 DMAC レジスタに複数の #pragma DMAC を記述する事はできません。
 (4) #pragma DMAC で指定された変数に対して、"&" (アドレス演算子)、"()" (関数呼び出し演算子)、"[]" (配列添字演算子)、"->" (間接メンバ演算子)を指定する事はできません。
 (5) #pragma DMAC で指定された変数は、volatile指定が無くても、volatile指定がされているものとして処理されます。

使用例

```
void _far *dda0;
#pragma DMAC dda0 DDA0

void func(void)
{
    unsigned char buff[10];

    dda0 = buff;
}
```

図B.73 #pragma DMAC宣言

#pragma INTCALL

ソフトウェア割り込み関数宣言機能

- 機能** ソフトウェア割り込み(int命令)で呼び出す関数を宣言します。
- 書式**
- (1) #pragma INTCALL△INT番号△アセンブラ関数名(レジスタ名,レジスタ名, ...)
 - (2) #pragma INTCALL△INT番号△C言語関数名()
- 解説** 指定したINT番号によってint命令を発行して、ソフトウェア割り込みにより関数の呼び出しを行いません。
- 規定**
- アセンブラ関数を宣言する場合
 - (1) #pragma INTCALL宣言を行う前には、必ずアセンブラ関数のプロトタイプ宣言を行ってください。プロトタイプ宣言を行わない場合は警告を出力し、この宣言を無効にします。
 - (2) プロトタイプ宣言では、以下の項目を守ってください。
 - (1) プロトタイプ宣言の引数の数と#pragma INTCALL宣言の引数の数が一致している必要があります。
 - (2) アセンブラ関数の引数に以下の型は宣言できません。
 - 構造体型
 - 共用体型
 - (3) アセンブラ関数のリターン値の型として以下の関数は宣言できません。
 - 構造体、共用体を返す関数
 - (3) 呼び出し時は、引き数として以下のレジスタを使用することができます。
 - double型、long long型 (64ビットレジスタ)
R3R1R2R0, R7R5R6R4, A1A0, A3A2
 - float型、long型、int型、far*(farポインタ) (32ビットレジスタ)
R2R0, R3R1, R6R4, R7R5, A0, A1, A2, A3
 - short型、int型(-fint_16オプション使用時) (16ビットレジスタ)
R0, R1, R2, R3, R4, R5, R6, R7
 - char型、_Bool型 (8ビットレジスタ)
R0L, R0H, R1L, R1H, R2L, R2H, R3L, R3H
 - レジスタ名を記述する際、大文字、小文字の区別はしません。
 - (4) INT番号は、10進数でのみ記述可能です。

#pragma INTCALL

ソフトウェア割り込み関数宣言機能

- 規定:
- C言語で実体を記述した関数を宣言する場合
 - (1) #pragma INTCALL宣言を行う前には、必ず関数のプロトタイプ宣言を行ってください。プロトタイプ宣言を行わない場合は警告を出力し、この宣言を無効にします。
 - (2) #pragma INTCALL宣言を行う関数の引数にレジスタ名を指定できません。
 - (3) プロトタイプ宣言では、以下の項目を守ってください。
 - (1) プロトタイプ宣言では、関数の呼び出し規則において、全ての引数がレジスタ渡しとなる関数のみ宣言できます。
 - (2) 関数のリターン値の型として以下の関数は宣言できません。
 - 構造体、共用体を返す関数
 - (4) INT番号は、10進数でのみ記述可能です。

使用例:

```
int    asm_func(unsigned long, unsigned short); ← アセンブラ関数のプロトタイプ宣言
#pragma INTCALL 25 asm_func(R2R0, R1)

void   main(void)
{
    int    i;
    long   l;

    i = 0x7FFD;
    l = 0x007F;

    asm_func( l, i ); ← アセンブラ関数の呼び出し
}
```

図B.74 #pragma INTCALL宣言の使用例 (1) (アセンブラ関数を宣言する場合)

```
int    c_func(unsigned int, unsigned int); ← C言語関数のプロトタイプ宣言
#pragma INTCALL 25 c_func(); ← 引数のレジスタ名を指定しないでください。

void   main(void)
{
    int    i, j;

    i = 0x7FFD;
    j = 0x007F;

    c_func( i, j ); ← C言語関数の呼び出し
}
```

図B.75 #pragma INTCALL宣言の使用例 (2) (C言語で実体を記述した関数を宣言する場合)

- 備考: 付属のスタートアップファイルをご使用になる場合は、vectorセクションの内容を変更して下さい。変更方法の詳細は、「スタートアッププログラムの準備」を参照してください。

#pragma INTERRUPT

割り込み関数の記述機能

- 機能** 割り込み処理関数の宣言を行います。
- 書式**
- (1) #pragma INTERRUPT△[B!/E!/F!/R!/V]△割り込み処理関数名
 - (2) #pragma INTERRUPT△[B!/E!/F!/R]△割り込みベクタ番号 割り込み処理関数名
 - (3) #pragma INTERRUPT△[B!/E!/F!/R]△割り込み処理関数名(vect=割り込みベクタ番号)
- 解説**
- (1) C言語で記述した割り込み処理関数を上記の書式で宣言することにより、関数の出入り口で以下の割り込みのための処理を行うコードを生成します。
 - 入り口処理では、マイコンのすべてのレジスタをスタックに退避します。
 - 出口処理では、退避したレジスタを復帰させて、REIT命令でリターンします。
 - (2) 宣言時に以下のスイッチを指定できます。
 - [B]
関数呼び出し時にレジスタをスタックに退避する代わりにレジスタバンクを切り換えることができます。これにより、高速な割り込み処理を実現することができます。
 - [F]
出口処理で、FREIT でリターンします。
 - [E]
割り込みに入った直後に多重割り込みを許可します。これにより、割り込みの応答性が向上します。
 - [R]
FLGレジスタの浮動小数点丸め演算モードを「最近値」に変更するコードを出力しません。
 - [V]
割り込み関数のベクタテーブルを生成するのみで、生成コードは変更しません。主に固定ベクタで使用します。
 - (3) 宣言時に割り込みベクタ番号を指定できます。
割り込みベクタ番号を設定してコンパイルすることで、可変ベクタテーブルを自動的に生成することができます。
ベクタ番号を指定せず、アセンブリ言語スタートアッププログラムを使用する場合は、「2.2.2 スタートアッププログラムのカスタマイズ」の「e. 割り込みベクタテーブルの設定」を参照してください。

#pragma INTERRUPT

割り込み関数の記述機能

- 規 定:
- (1) 引数を持つ割り込み処理関数を記述した場合、コンパイル時に警告を出力します。
 - (2) 戻り値を返す割り込み処理関数を記述した場合、コンパイル時に警告を出力します。関数の戻り値は、必ずvoid型であるように宣言してください。
 - (3) #pragma INTERRUPT宣言以降に関数の実体を定義した関数のみ有効となります。
 - (4) 関数名以外を指定した場合、何の効果も及ぼしません。
 - (5) #pragma INTERRUPT宣言の二重定義はエラーとなりません。
 - (6) スイッチ/Eと/Bは同時に指定できません。
 - (7) 同じ割り込み処理関数に、異なる割り込みベクタ番号を記述した場合は、後に宣言されたベクタ番号が有効になります。
 - (8) /Nとその他のスイッチは同時に使用できません。

```
#pragma INTERRUPT intr(vect=10)
#pragma INTERRUPT intr(vect=20) /* 割り込みベクタ番号 20 が有効 */
```

図B.76 異なる割り込みベクタ番号の記述例

使用例:

```
extern int  int_counter;

#pragma INTERRUPT /B i_func

void      i_func(void)
{
    int_counter += 1;
}
```

図B.77 #pragma INTERRUPT宣言の使用例

- 備 考:
- (1) 付属のスタートアップファイルをご使用になる場合は、vectorセクションの内容を変更して下さい。変更方法の詳細は、「スタートアッププログラムの準備」を参照してください。
 - (2) 裏レジスタを使用する場合、割り込みのネストにより裏レジスタが破壊されないように注意してください。

#pragma PARAMETER**レジスタ渡しのアセンブラ関数宣言機能**

- 機能** 引数をレジスタに格納して渡すアセンブラ関数を宣言します。
- 書式** #pragma PARAMETER Δアセンブラ関数名(レジスタ名, レジスタ名, ..)
- 解説** アセンブラ関数を呼び出すときに、その引数を以下のレジスタに格納して渡すことを宣言します。
- double型、long long型 (64ビットレジスタ)
R3R1R2R0, R7R5R7R4, A1A0, A3A2
 - float型、long型、int型、far *{farポインタ} (32ビットレジスタ)
R2R0, R3R1, R6R4, R7R5, A0, A1, A2, A3
 - short型、int型(-fint_16オプション使用時) (16ビットレジスタ)
R0, R1, R2, R3, R4, R5, R6, R7
 - char型、_Bool型 (8ビットレジスタ)
R0L, R0H, R1L, R1H, R2L, R2H, R3L, R3H
 - レジスタ名を記述する際、大文字、小文字の区別はしません。
 - 構造体型、共用体型は、宣言できません。
- 規定**
- (1) #pragma PARAMETER宣言を行う前には、必ずアセンブラ関数のプロトタイプ宣言を行ってください。プロトタイプ宣言を行わない場合は警告を出力し、この宣言を無効にします。
 - (2) プロトタイプ宣言では、以下の項目を守ってください。
 - (1) プロトタイプ宣言の引数の数と#pragma PARAMETER宣言の引数の数が一致している必要があります。
 - (2) アセンブラ関数の引数の型として、以下の型は宣言できません。
 - 構造体型
 - 共用体型
 - (3) アセンブラ関数のリターン値の型として以下の関数は宣言できません。
 - 構造体、共用体を返す関数

#pragma PARAMETER

レジスタ渡しのアセンブラ関数宣言機能

使用例:

```
short  asm_func(short, short); ← アセンブラ関数のプロトタイプ宣言
#pragma PARAMETER asm_func(R0, R1)

void  main(void)
{
    short  i, j;

    i = 0x7FFD;
    j = 0x007F;

    asm_func( i, j ); ← アセンブラ関数の呼び出し
}
```

図B.78 #pragma PARAMETER宣言の使用例

B.7.4 その他の拡張機能の使用法

本コンパイラは、前述以外に以下の拡張機能を持っています。

#pragma __ASMMACRO

アセンブラマクロ関数宣言機能

機能: アセンブラのマクロで定義した関数を宣言します。

書式: #pragma __ASMMACRO 関数名(レジスタ名,...)

- 規定:**
- (1) 本機能による宣言を行う前に、関数のプロトタイプ宣言を行ってください。アセンブラマクロ関数は、必ず **static** 宣言してください。
 - (2) 引数のない関数は宣言できません。引数はレジスタ渡しになります。引数の型と合致するレジスタを指定してください(#pragma PARAMETER に準じます)。
 - (3) 宣言した関数名の先頭にアンダスコア(_)を付加したマクロ名で、アセンブラマクロを定義してください。
 - (4) 戻り値の返し方は、関数呼出し規則に従い、以下のようになります。集合体(構造体・共用体)を戻り値とすることはできません。

_Bool型、char型:	R0L	float型:	R2R0
int(-fint_16使用時)、short型:	R0	double型:	A1A0
int(-fint_16未使用時)、long 型:	R2R0	long long型:	A1A0
ポインタ型:	A0		

- (5) アセンブラマクロ内で内容が変更されるレジスタは、アセンブラマクロの先頭で退避して、復帰直前に復帰してください(戻り値の格納レジスタの退避・復帰は不要です)。

使用例:

```
static short max_w( short, short ); /* 必ず static にしてください */

#pragma __ASMMACRO max_w( R0, R2 )
#pragma ASM
_max_w .macro
        max.w    R2,R0    ; 戻り値はR2R0 で返されます
        .endm
#pragma ENDASM

short    s;

void    test_func( void )
{
        s= max_w( 2, 3 );
}
```

図B.79 #pragma __ASMMACRO 記述例

#pragma ASM～#pragma ENDASM

インラインアセンブラ指定機能

- 機能** アセンブリ言語で記述を行なう領域を指定します。
- 書式** #pragma ASM
アセンブリ言語記述
#pragma ENDASM
- 解説** #pragma ASMと#pragma ENDASMの間の領域を直接、アセンブリ言語ファイルにそのまま出力します。
#pragma ASM を記述する際は、必ず#pragma ENDASMを組み合わせて記述してください。#pragma ASMと対になる#pragma ENDASMがない場合はNC100は処理を中断します。
- 規定**
- (1) アセンブリ言語記述において、レジスタの内容を破壊する記述をしないでください。レジスタの内容を破壊する記述をする場合は、push命令・pop命令を使用して、レジスタ内容の退避・復帰を行ってください。
 - (2) "#pragma ASM～#pragma ENDASM"内では、引数およびauto変数を参照しないでください。
 - (3) "#pragma ASM～#pragma ENDASM"内でフローに影響を与えるようなブランチ(条件ブランチ含む)を記述しないようにしてください。

使用例

```
void func(void)
{
    int i, j;

    for(i=0; i < 10;i++){
        func2();
    }

#pragma ASM
LOOP1:
    FCLR I
    MOV.W #0FFH,R0
    :
    (省略)
    :
    FSET I
#pragma ENDASM
}
```

この領域をアセンブリ言語ファイルにそのまま出力します。

図B.80 #pragma ASM(ENDASM)記述例

- 補足** #pragma ASM～#pragma ENDASMまでに記述されたアセンブリ言語プログラムは、C言語プリプロセッサの処理対象となります。

#pragma JSRA

関数呼び出し命令指定機能

機能 関数をJSR.A命令により呼び出します。

書式 #pragma JSRA△関数名

解説 #pragma JSRA で宣言した関数は、JSR.A命令固定で関数呼び出しを行います。
コンパイルオプション"-fJSRW"で生成したコードで、リンク時にエラーとなる関数は、#pragma JSRA
を指定する事によりエラーを回避できます。

規定 コンパイルオプション"-fJSRW"を指定しない場合は、効果がありません。

使用例

```
extern void    func(int i);
#pragma JSRA func()

void    main(void)
{
    func(1);
}
```

図B.81 #pragma JSRA 記述例

#pragma JSRW

関数呼び出し命令指定機能

機能: 関数をJSR.W命令により呼び出します。

書式: #pragma JSRW△関数名

解説: 任意の関数から、同一ファイルに実体の定義がない関数を呼び出す場合、通常ではJSR.A命令で呼び出します。#pragma JSRW で宣言した関数は、JSR.W命令固定で関数呼び出しを行います。この拡張機能によりROM容量を削減できます。

規定:

- (1) static関数には、#pragma JSRWを指定しないでください。
- (2) 関数呼び出し時に#pragma JSRW で宣言した関数に届かない時は、リンク時にエラーとなります。この場合は、本宣言を行わないでください。

使用例:

```
#pragma JSRW func()

void main(void)
{
    func(1);
}
```

図B.82 #pragma JSRW 記述例

備考: #pragma JSRWは、関数の直接呼び出しの時にのみ有効です。間接呼び出しの時は、効果がありません。

#pragma PAGE

.PAGE疑似命令出力機能

機能 アセンブラで出力するリストファイルでの改ページ位置を宣言します。

書式 #pragma PAGE

解説 ソースファイル中に#pragma PAGEを記述した場合、コンパイラが出力するアセンブリ言語ファイルに、指示命令.PAGEを出力します。アセンブラによりアセンブラリスティングファイルを出力する場合に、改ページ指定を行うことができます。

- 規定**
- (1) アセンブラ疑似命令.PAGEのヘッダに指定する文字列の指定はできません。
 - (2) auto変数の宣言中に#pragma PAGEを記述できません。

使用例

```
void    func(void)
{
    int    i, j;

    for(i=0; i < 10;i++){
        func2();
    }
#pragma PAGE
    i++;
}
```

図B.83 #pragma PAGE 記述例

B.8 アセンブラマクロ関数

B.8.1 アセンブラマクロ関数の概要

本コンパイラでは、アセンブラ命令の一部をC言語の関数として記述することができます。

この機能を使用することにより、特定のアセンブラの命令を直接的にC言語のプログラム上に記述できるので、プログラムのチューンアップが行いやすくなります。

B.8.2 アセンブラマクロ関数の記述例

アセンブラマクロ関数は【図B.84】のように、C言語プログラム中にC言語の関数と同じ書式で記述することができます。アセンブラマクロ関数機能を使用する場合は、必ず`asmmacro.h`をインクルードしてください。

```
#include <asmmacro.h>          /* アセンブラマクロ関数の定義ファイルをインクルード */
long    l;
char    a[20];
char    b[20];

void    func(void)
{
    l = rmpa_b(0,19,a,b);      /* アセンブラマクロ関数(rmpa 命令) */
}
```

図B.84 アセンブラマクロ関数の記述例

B.8.3 アセンブラマクロ関数で記述可能な命令

アセンブラマクロ関数で記述可能なアセンブラ命令と、アセンブラマクロ関数としての機能と書式を示します。

ABS

機能: val の絶対値を返します。

書式: `#include <asmmacro.h>`

```
/* 8bit での演算の場合 */
static signed char abs_b( signed char val );

/* 16bit 出の演算の場合 */
static short int abs_w( short int val );

/* 32bit での演算の場合 */
static long int abs_l( long int val );
```

MAX

機能: val1 と val2 を比較して大きい方の値を返します。

書式: `#include <asmmacro.h>`

```
/* 8bit での演算の場合 */
static signed char max_b( signed char val1, signed char val2 );

/* 16bit での演算の場合 */
static short int max_w( short int val1, short int val2 );

/* 32bit での演算の場合 */
static long int max_l( long int val1, long int val2 );
```

MIN

機能: val1 と val2 を比較して小さい方の値を返します。

書式: #include <asmmacro.h>

```
/* 8bit での演算の場合 */
static signed char min_b( signed char val1, signed char val2 );

/* 16bit での演算の場合 */
static short int min_w( short int val1, short int val2 );

/* 32bit での演算の場合 */
static long int min_l( long int val1, long int val2 );
```

RMPA

機能: 初期値 : init、回数 : count、乗数の格納されている先頭アドレスをそれぞれ p1、p2 として、積和演算を行い結果を返します。

書式: #include <asmmacro.h>

```
/* 8bit での演算の場合 */
static long long rmpa_b( long long init, unsigned long int count,
                        const signed char _far *p1, const signed char _far *p2 );

/* 16bit での演算の場合 */
static long long rmpa_w( long long init, unsigned long int count,
                        const short int _far *p1, const short int _far *p2 );

/* 32bit での演算の場合 */
static long long rmpa_l( long long init, unsigned long int count,
                        const long int _far *p1, const long int _far *p2 );
```

SIN

機能: p1 で示される固定の転送元番地から、p2 で示される転送先番地に count 回数分、アドレスの加算方向へストリング転送を行います。戻り値はありません。

書式:

```
#include <asmmacro.h>

/* 8bit での演算の場合 */
static void sin_b( volatile const void _far *p1, void _far *p2,
                  unsigned long int count );

/* 16bit での演算の場合 */
static void sin_w( volatile const void _far *p1, void _far *p2,
                  unsigned long int count );

/* 32bit での演算の場合 */
static void sin_l( volatile const void _far *p1, void _far *p2,
                  unsigned long int count );
```

SMOVB

機能: p1 で示される転送元番地から、p2 で示される転送先番地に count 回数分、アドレスの減算方向へストリング転送を行います。戻り値はありません。

書式:

```
#include <asmmacro.h>

/* 8bit での演算の場合 */
static void smovb_b( const void _far *p1, void _far *p2,
                    unsigned long int count );

/* 16bit での演算の場合 */
static void smovb_w( const void _far *p1, void _far *p2,
                    unsigned long int count );

/* 16bit での演算の場合 */
static void smovb_l( const void _far *p1, void _far *p2,
                    unsigned long int count );
```

SMOVF

機能: p1 で示される転送元番地から、p2 で示される転送先番地に count 回数分、アドレスの加算方向へストリング転送を行います。戻り値はありません。

書式: #include <asmmacro.h>

```
/* 8bit での演算の場合 */
static void smovf_b( const void _far *p1, void _far *p2,
                    unsigned long int count );

/* 16bit での演算の場合 */
static void smovf_w( const void _far *p1, void _far *p2,
                    unsigned long int count );

/* 16bit での演算の場合 */
static void smovf_l( const void _far *p1, void _far *p2,
                    unsigned long int count );
```

SMOVU

機能: p1 で示される転送元番地から、p2 で示される転送先番地に、アドレスの加算方向へ零が検出されるまで、ストリング転送を行います。戻り値はありません。

書式: #include <asmmacro.h>

```
/* 8bit での演算の場合 */
static void smovu_b( const void _far *p1, void _far *p2 );

/* 16bit での演算の場合 */
static void smovu_w( const void _far *p1, void _far *p2 );
```

SOUT

機能: p1 で示される転送元番地からアドレスの加算方向へ、p2 で示される転送先番地に count 回数分ストリング転送を行います。戻り値はありません。

書式: #include <asmmacro.h>

```
/* 8bit での演算の場合 */
static void sout_b( const void _far *p1, volatile void _far *p2,
                   unsigned long int count );

/* 16bit での演算の場合 */
static void sout_w( const void _far *p1, volatile void _far *p2,
                   unsigned long int count );

/* 32bit での演算の場合 */
static void sout_l( const void _far *p1, volatile void _far *p2,
                   unsigned long int count );
```

SSTR

機能: val をストアするデータ、p を転送するアドレス、count を転送回数としてストリングストアを行います。戻り値はありません。

書式: #include <asmmacro.h>

```
/* 8bit での演算の場合 */
static void sstr_b( unsigned char val, void _far *p,
                   unsigned long int count );

/* 16bit での演算の場合 */
static void sstr_w( unsigned short int val, void _far *p,
                   unsigned long int count );

/* 32bit での演算の場合 */
static void sstr_l( unsigned long int val, void _far *p,
                   unsigned long int count );
```

SUNTIL

機能: count で指定される回数、from で示される比較先番地からアドレスの加算方向に、val と一致するデータが現れるまで検索を行います。

書式: `#include <asmmacro.h>`

```
/* 8bit での演算の場合 */
static void _far *suntil_b( unsigned char val, const void _far *from,
                           unsigned long int count );

/* 16bit での演算の場合 */
static void _far *suntil_w( unsigned short int val, const void _far *from,
                           unsigned long int count );

/* 32bit での演算の場合 */
static void _far *suntil_l( unsigned long int val, const void _far *from,
                           unsigned long int count );
```

SWHILE

機能: count で指定される回数、from で示される比較先番地からアドレスの加算方向に、val と一致しないデータが現れるまで検索を行います。

書式: `#include <asmmacro.h>`

```
/* 8bit での演算の場合 */
static void _far *swhile_b( unsigned char val, const void _far *from,
                           unsigned long int count );

/* 16bit での演算の場合 */
static void _far *swhile_w( unsigned short int val, const void _far *from,
                           unsigned long int count );

/* 32bit での演算の場合 */
static void _far *swhile_l( unsigned long int val, const void _far *from,
                           unsigned long int count );
```

付録C C 言語仕様概要

C 言語仕様は、標準的な C 言語の仕様に加え、ROM 化を容易に行うための拡張機能を持っています。

C.1 性能仕様

C.1.1 標準仕様概要

本コンパイラは、R32C/100 シリーズをターゲットにしたクロス環境の C コンパイラです。言語仕様のには、以下の R32C/100 シリーズのハードウェア的な仕様、及び ROM 化を容易にするために用意した拡張機能を除けば、標準的なフルセットの C 言語とほぼ同様の仕様を持っています。

- ROM化のための拡張機能(near/far修飾子、asm関数等)
- 標準ライブラリ関数の中で浮動小数点ライブラリやホストマシンに依存した内容

C.1.2 性能概要

以下に本コンパイラの性能の概要を示します。

(a) 測定環境

性能測定時のPCの標準環境を【表C.1】に示す条件として想定しています。

表C.1 PC標準環境

項目	PCの種類	OSのバージョン
パーソナルコンピュータの環境	IBM PC/AT 互換機	Windows XP, Windows Me, Windows 98, Windows 2000, Windows NT 4.0

(b) C 言語ソースファイル記述仕様

【表C.2】に本コンパイラのC言語ソースファイル記述に関する仕様を示します。なお、実測が不可能な一部の仕様は計算による予想値を示しています。

表C.2 C言語ソースファイル記述仕様

項目	仕様
ソースファイルでの1行の文字数	改行コードを含む512バイト(文字)
ソースファイルでの行数	最大65535行

(c) 仕様

【表C.3】に本コンパイラの仕様を示します。なお、実測が不可能な一部の仕様は計算による予想値を示しています。

表C.3 仕様

項目	仕様
指定可能なファイル数	メモリの容量に依存します
ファイル名の長さ	OS に依存します
起動オプション-D で指定可能なマクロ名の総数	メモリの容量に依存します
起動オプション-I で指定可能なディレクトリ数	最大 256
起動オプション-as100 で引き渡し可能なパラメータ数	メモリの容量に依存します
起動オプション-ln100 で引き渡し可能なパラメータ数	メモリの容量に依存します
複文、繰り返し制御構造、及び選択制御構造に対するネスト数	メモリの容量に依存します
条件コンパイルにおけるネスト数	メモリの容量に依存します
宣言中の基本型を修飾するポインタ、配列、及び関数宣言子の数	メモリの容量に依存します
関数定義の数	メモリの容量に依存します
1つのブロック中におけるブロック有効範囲を持つ識別子の数	メモリの容量に依存します
1つのソースファイル中で同時に定義され得るマクロ識別子の数	メモリの容量に依存します
マクロ名の置き換えの数	メモリの容量に依存します
入力プログラムにおける論理ソース行数	メモリの容量に依存します
#include ファイルに対するネスト数	最大 40
1つの switch 文中における case 名札の数 (switch 文のネストがない場合)	メモリの容量に依存します
#if、#elif 文で指定できる演算子、被演算子の合計数	メモリの容量に依存します
関数 1 個あたりで確保可能なスタックフレーム容量(バイト数)	最大 64K
#pragma ADDRESS で定義可能な変数の数	メモリの容量に依存します
括弧のネスト数	メモリの容量に依存します
初期化式付きの変数定義を行う場合の定義可能な初期値の数	メモリの容量に依存します
修飾宣言子のネスト数	YACC スタックに依存
宣言子の括弧によるネスト数	YACC スタックに依存
演算子の括弧によるネスト数	YACC スタックに依存
1つの内部識別子又はマクロ名で意味をもつ文字数	メモリの容量に依存します 最大200
1つの外部識別子で意味をもつ文字数	メモリの容量に依存します 最大200
1ソースファイル中の外部識別子の数	メモリの容量に依存します
1ブロックでブロックスコープを持つ識別子	メモリの容量に依存します
1ソースファイル中のマクロ数	メモリの容量に依存します
1つの関数、関数呼び出しのパラメータ数	メモリの容量に依存します
1つのマクロ定義、マクロ呼び出しのパラメータ数	最大 31
結合後の文字列リテラル内の文字数	メモリの容量に依存します
1つのオブジェクトサイズ(バイト数)	メモリの容量に依存します
1つの構造体/共用体のメンバ数	メモリの容量に依存します
1つの列挙中の列挙定数の数	メモリの容量に依存します
1つの struct 宣言リスト中の構造体/共用体のネスト数	メモリの容量に依存します
1つの文字列の文字数	OS に依存します
1ファイルの行数	メモリの容量に依存します

C.2 基本言語仕様

本コンパイラの言語仕様を基本的な言語仕様と併せて説明します。

C.2.1 文法

文法の字句要素について説明します。本コンパイラは以下のものを字句(トークン)として処理します。

- キーワード
- 識別子
- 定数
- 文字リテラル
- 演算子
- 区切り子
- 注釈

(a) キーワード

本コンパイラは【表C.4】のものをキーワードとして解釈します。

表C.4 キーワード一覧表

<code>_asm</code>	<code>_far</code>	<code>_near</code>	<code>asm</code>	<code>auto</code>
<code>_Bool</code>	<code>_inline</code>	<code>break</code>	<code>case</code>	<code>char</code>
<code>const</code>	<code>continue</code>	<code>default</code>	<code>do</code>	<code>double</code>
<code>else</code>	<code>enum</code>	<code>extern</code>	<code>far</code>	<code>float</code>
<code>for</code>	<code>goto</code>	<code>if</code>	<code>inline</code>	<code>int</code>
<code>long</code>	<code>near</code>	<code>register</code>	<code>restrict</code>	<code>return</code>
<code>short</code>	<code>signed</code>	<code>sizeof</code>	<code>static</code>	<code>struct</code>
<code>switch</code>	<code>typedef</code>	<code>union</code>	<code>unsigned</code>	<code>void</code>
<code>volatile</code>	<code>while</code>			

(b) 識別子

識別子は、以下の要素で構成されます。

- 1文字目は英字又はアンダースコア(A~Z、a~z、_)
- 2文字目以降は英数字又はアンダースコア(A~Z、a~z、0~9、_)

識別子名は最大 200 文字まで記述できます。ただし、日本語文字は識別子として記述できません。

(c) 定数

定数は以下に示す 3 種類のタイプがあります。

- 整数定数
- 浮動小数点定数
- 文字定数

(1) 整数定数

整数定数は、10 進数のほか、8 進数、16 進数を指定することができます。各進数の書式を【表C.5】に示します。

表C.5 整数定数の記述法

進数	記述法	構成	記述例
10 進数	なにも付けない	0123456789	15
8 進数	0(ゼロ)で始まる	01234567	017
16 進数	0X 又は 0x で始まる	0123456789ABCDEF 0123456789abcdef	0XF 又は 0xf

整数定数の型は、その値の大小により以下の順で決定されます。

- 8進数、16進数
signed int型 → unsigned int型 → signed long型 → unsigned long型 → signed long long型
→ unsigned long long型
- 10進数
signed int型 → signed long型 → signed long long型

また、接尾詞 U 又は u、L 又は l、LL 又は ll を付加した場合、以下のように扱われます。

- (1) 符号無し定数
符号無し定数は、定数値の後にU又はuを付加して記述します。型は値により以下の順で決定されます。
unsigned int型 → unsigned long型 → unsigned long long型
- (2) long型定数
long型定数は、定数値の後にL又はlを付加して記述します。型は値により以下の順で決定されます。
 - 8進数、16進数
signed long型 → unsigned long型 → signed long long型 → unsigned long long型
 - 10進数
signed long型 → signed long long型
- (3) long long型定数
long long型定数は、定数値の後にLL又はllを付加して記述します。型は値により以下の順で決定されます。
 - 8進数、16進数
signed long long型 → unsigned long long型
 - 10進数
signed long long型

(2) 浮動小数点定数

浮動小数点定数は、定数値の後になにも付加しない場合、double 型として扱われます。float 型として扱う場合は、定数値の後に F 又は f を付加して記述します。また、L 又は l を付加した場合は、long double 型として扱われます。

(3) 文字定数

文字定数は通常、'文字'のようにシングルクォーテーションの中に文字を記述して表現します。文字の中には以下のような拡張表記(エスケープ系列/トライグラフ系列)が使用できます。16 進数と 8 進数の表現は、¥x の後に 16 進数を続けると 16 進数に、¥ の後に 8 進数を続けると 8 進数となります。

表C.6 拡張表記一覧表

表記	内容(エスケープ系列)	表記	内容(トライグラフ系列)
¥	シングルクォーテーション	¥定数値	8進数
¥"	ダブルクォーテーション	¥x 定数値	16進数
¥¥	逆スラッシュ	??(文字 [を表します
¥?	疑問符	??/	文字 ¥ を表します
¥a	ベル文字	??)	文字] を表します
¥b	バックスペース	??'	文字 ^ を表します
¥f	改ページ	??<	文字 { を表します
¥n	改行	??!	文字 ! を表します
¥r	復帰	??>	文字 } を表します
¥t	水平タブ	??-	文字 ` を表します
¥v	垂直タブ	??=	文字 # を表します

(d) 文字リテラル

文字リテラルは、"文字列"のようにダブルクォーテーションの中に文字列を記述して表現します。文字リテラルにも【表C.6】に示した文字定数と同じ拡張表記が使用できます。

(e) 演算子

本コンパイラは、【表C.7】に示すものを演算子として解釈します。

表C.7 演算子一覧表

単項演算子	++	論理演算子	&&
	--		
	-		!
二項演算子	+	条件演算子	?:
	-	カンマ演算子	,
	*	アドレス演算子	&
	/	ポインタ演算子	*
	%	ビット演算子	<<
代入演算子	=		>>
	+=		&
	-=		!
	*=		^
	/=		~
	%=		&=
関係演算子	>		!=
	<		^=
	>=		<<=
	<=		>>=
	==	sizeof 演算子	sizeof
	!=		

(f) 区切り子

本コンパイラは、以下に示すものを区切り子として解釈します。

- {
- :
- ,
- }
- ;
-

(g) 注釈

注釈は、/*で始まり*/で終了します。注釈のネストはできません。

注釈は、// で始まり行末で終了します。

C.2.2 型

(a) データ型

本コンパイラでは、以下に示すデータ型をサポートしています。

- 文字型
- 構造体
- 列挙型
- 浮動小数点型
- 整数型
- 共用体
- void型

(b) 型修飾子

本コンパイラでは、以下に示すものを型修飾子として解釈します。

- const
- restrict
- far
- volatile
- near

(c) データ型とサイズ

各データ型に対応するビットサイズを【表C.8】に示します。

表C.8 データ型とビットサイズ

型名	符号の有無	ビットサイズ	表現できる数値
_Bool	なし	8	0, 1
char unsigned char	なし	8	0~255
signed char	有り	8	-128~127
int short signed int signed short	有り	16	-32768~32767
unsigned int unsigned short	なし	16	0~65535
int long signed int signed long	有り	32	-2147483648~2147483647
unsigned int unsigned long	なし	32	0~4294967295
long long signed long long	有り	64	-9223372036854775808~9223372036854775807
unsigned long long	なし	64	18446744073709551615
float	有り	32	1.17549435e-38F~3.40282347e+38F
double long double	有り	64	2.2250738585072014e-308~ 1.7976931348623157e+308
far ポインタ	なし	32	0~0xFFFFFFFF

- _Bool型の符号指定はできません。
- char型は、符号指定がない場合unsigned char型と解釈します。
- int型、short型は、符号指定がない場合signed int型、signed short型と解釈します。
- long型は、符号指定がない場合signed long型と解釈します。
- long long型は、符号指定がない場合signed long long型と解釈します。
- 構造体のビットフィールドメンバで符号指定がない型は、符号なしと解釈します。
- long long型のビットフィールドは使用できません。
- int型は32ビットで扱います。ただし、コンパイルオプション“-fint16(-fint_16)”を指定した場合は16ビットで扱います。

C.2.3 式

【表C.9】に式の種類と式の構成要素の関係を示します。

表C.9 式の種類と構成要素

一次式	識別子	関係式	式
	定数		式 < 式
	文字リテラル		式 > 式
	(式)		式 <= 式
	一次式		式 >= 式
後置式	後置式[式]	等価式	式 == 式
	後置式(引数の並び, ...)		式 != 式
	後置式. 識別子	ビット単位の AND 式	式 & 式
	後置式->識別子	ビット単位の排他的 OR 式	式 ^ 式
	後置式++	ビット単位の OR 式	式 式
	後置式--	論理 AND 式	式 && 式
	後置式	論理 OR 式	式 式
単項式	++単項式	条件式	式 ? 式 : 式
	--単項式		代入式
	単項演算子 キャスト式	単項式 -= 式	
	sizeof 単項式	単項式 *= 式	
	sizeof(型名)	単項式 /= 式	
単項式	単項式 %= 式		
キャスト式	(型名)キャスト式		単項式 <<= 式
	キャスト式		単項式 >>= 式
式	式 * 式		単項式 &= 式
	式 / 式		単項式 = 式
	式 % 式		単項式 ^= 式
加減式	式 + 式		代入式
	式 - 式	コンマ演算子	
ビット単位のシフト式	式 << 式		
	式 >> 式		

C.2.4 宣言

宣言には以下に示す2種類のタイプがあります。

- 変数宣言
- 関数宣言

(a) 変数宣言

変数の宣言は、【図C.1】に示す書式で記述します。

```
記憶クラス指定子△型宣言子△宣言指定子△初期化式;
```

図C.1 変数の宣言書式

(1) 記憶クラス指定子

本コンパイラでは、以下の記憶クラス指定子をサポートしています。

- extern
- static
- typedef
- auto
- register

(2) 型宣言子

本コンパイラでは、以下の型宣言子をサポートしています。

- _Bool
- int
- long
- float
- unsigned
- struct
- enum
- char
- short
- long long
- double
- signed
- union

(3) 宣言指定子

本コンパイラでは、【図C.2】に示す書式で宣言指定子を記述します。

宣言子:	ポインタ opt 宣言子 2
宣言子 2 :	識別子 (宣言子)
	宣言子 2 [定数式 opt]
	宣言子 2 (仮引数の並び opt)

※配列の個数を示す定数式は最初の配列のみ省略できます。
 ※opt はオプション部であることを示します。

図C.2 宣言指定子の書式

(4) 初期化式

本コンパイラでは、初期化式に【図C.3】に示す初期値を記述できます。

整数型:	定数
整数型配列:	定数、定数 ...
文字型:	定数
文字型配列:	文字リテラル定数、定数 ...
ポインタ型:	文字リテラル
ポインタ配列:	文字リテラル、文字リテラル ...

図C.3 初期化式に記述できる初期値

(b) 関数宣言

関数の宣言は、【図C.4】に示す書式で記述します。

関数宣言(定義): 記憶クラス指定子△型宣言子△宣言指定子△プログラム本体 関数宣言(プロトタイプ宣言): 記憶クラス指定子△型宣言子△宣言指定子;

図C.4 関数の宣言書式

(1) 記憶クラス指定子

本コンパイラでは、以下の記憶クラス指定子をサポートしています。

- extern
- static

(2) 型宣言子

本コンパイラは、以下の型宣言子をサポートしています。

- _Bool
- int
- long
- float
- unsigned
- struct
- enum
- char
- short
- long long
- double
- signed
- union

(3) 宣言指定子

本コンパイラでは、【図C.5】に示す書式で宣言指定子を記述します。

宣言子: ポインタ opt 宣言子 2 宣言子 2: 識別子(仮引数の並び opt) (宣言子) 宣言子[定数式 opt] 宣言子(仮引数の並び opt) ※配列の個数を示す定数式は最初の配列のみ省略できます。 ※opt はオプション部であることを示します。 ※プロトタイプ宣言の場合は仮引数の並びでなく、型宣言子の並びとなります。

図C.5 宣言指定子の書式

(4) プログラム本体

プログラム本体は、【図C.6】に示す書式で記述します。

変数宣言子の並び_{opt} 複文

※プロトタイプ宣言の場合はプログラム本体は無く、セミコロンで終了します。

※_{opt} はオプション部であることを示します。

図C.6 プログラム本体の書式

C.2.5 文

本コンパイラでは、以下の文をサポートしています。

- 名札付き文
- 式/空文
- 繰り返し文
- アセンブリ言語記述文
- 複文
- 選択文
- 分岐文

(a) 名札付き文

名札付き文は、【図C.7】に示す書式で記述します。

識別子:	文
case 定数:	文
default:	文

図C.7 名札付き文の書式

(b) 複文

複文は、【図C.8】に示す書式で記述します。

{ 宣言の並び_{opt} 文の並び_{opt} }

※_{opt} はオプション部であることを示します。

図C.8 複文の書式

(c) 式/空文

式及び空文は、【図C.9】に示す書式で記述します。

```
式 :  
式 ;  
空文 :  
;
```

図C.9 式/空文の書式

(d) 選択文

選択文は、【図C.10】に示す書式で記述します。

```
if( 式 )文  
if( 式 )文 else 文  
switch( 式 )文
```

図C.10 選択文の書式

(e) 繰り返し文

繰り返し文は、【図C.11】に示す書式で記述します。

```
while( 式 )文 ;  
do 文 while ( 式 );  
for( 式 opt ; 式 opt ; 式 opt )文 ;  
  
※opt はオプション部であることを示します。
```

図C.11 繰り返し文の書式

(f) 分岐文

分岐文は、【図C.12】に示す書式で記述します。

```
goto 識別子 ;  
continue ;  
break ;  
return 式 opt ;  
  
※opt はオプション部であることを示します。
```

図C.12 分岐文の書式

(g) アセンブリ言語記述文

アセンブリ言語記述文は、【図C.13】に示す書式で記述します。

```
asm( "文字列" );
```

文字列： アセンブリ言語ステートメント

図C.13 アセンブリ言語記述文の書式

C.3 プリプロセスコマンド

プリプロセスコマンドは、#で始まるコマンドによりプリプロセッサ `cpp100` で処理されます。プリプロセスコマンドの仕様を説明します。

C.3.1 プリプロセスコマンドの機能別一覧

本コンパイラは、【表C.10】に示すプリプロセスコマンドを用意しています。

表C.10 プリプロセスコマンド一覧表

コマンド名	機能
<code>#assert</code>	定数式が偽のときに警告を出力します
<code>#define</code>	マクロを定義します
<code>#elif</code>	条件コンパイルを行います
<code>#else</code>	条件コンパイルを行います
<code>#endif</code>	条件コンパイルを行います
<code>#error</code>	メッセージを標準出力に出力し処理を中断します
<code>#ifndef if</code>	条件コンパイルを行います
<code>#ifdef</code>	条件コンパイルを行います
<code>#ifndef</code>	条件コンパイルを行います
<code>#include</code>	指定したファイルを取り込みます
<code>#line</code>	ファイルの行番号を指定します
<code>#pragma</code>	NC100の拡張機能の処理を指示します
<code>#undef</code>	マクロを未定義にします

C.3.2 プリプロセスコマンドリファレンス

以降に本コンパイラのプリプロセスコマンドの詳細仕様を説明します。

`#assert`

機能: 定数式が偽のときに警告を出力します。

書式: `#assert`△定数式

解説: 定数式の結果が0(ゼロ)の場合、以下の警告を發します。ただし、コンパイルはそのまま続行されます。

```
[Warning(cpp100):x.c, line xx]assertion warning
```

#define

機能 マクロを定義します。

書式 (1) `#define`△識別子△字句列
 (2) `#define`△識別子(識別子の並び)△字句列

解説 (1) 識別子をマクロとして定義します。
 (2) 識別子をマクロとして定義します。この書式では、最初の識別子と左括弧('('との間にスペース及びタブを入れないでください。

- 下記の記述をした場合、識別子は空白文字に置換されます。

```
#define SYMBOL
```

- マクロで関数を定義した場合、定義文中に逆スラッシュ又は'¥'を挿入することにより複数行にわたる記述が可能になります。
- 以下の4つの識別子はコンパイラの予約語です。

```
__FILE__ ..... ソースファイルの名前
__LINE__ ..... 現在のソースファイルの行番号
__DATE__ ..... コンパイルの日付(形式:mm dd yyyy)
__TIME__ ..... コンパイルの時間(形式:hh:mm:ss)
```

また、NC100では予め以下のマクロ(プリデファインドマクロ)が定義されています。

```
R32C100
NC100
__INT_16__ (コンパイルオプション-fI16 (-fint_16) 使用時に定義されます)
__CHAR_SIGNED__ (コンパイルオプション-fSC (-fsigned_char) 使用時に定義されます)
```

- 字句列に対して、文字列化演算子#及び字句結合演算子##を下記のように使用できます。

```
#define debug(s,t) printf("x"#s" = %d x"#t" = %d",x ## s,x ## t)
この識別子に引数を debug(1, 2)と与えたときは以下のように解釈されます。

#define debug(s,t) printf("x1 = %d x2 = %d", x1,x2)
```

- マクロの定義は下記のようにネスト(入れ子)することができます。ネストレベルは最大20です。

```
#define XYZ1 100
#define XYZ2 XYZ1
      :
      (省略)
      :
#define XYZ20 XYZ19
```

#error

機能 メッセージを標準出力に出力し処理を中断します。

書 式: #error△文字列

- 解 説
- コンパイルを中断します。
 - 字句列がある場合、その文字列を標準出力に出力します。

#if～#elif～#else～#endif

機能 条件コンパイルを行います(式が真であるか否かを調べます)。

書 式: #if△定数式
 :
 #elif△定数式
 :
 #else
 :
 #endif

- 解 説
- #ifと#elifは、定数の値が真(0でない)の場合、後に続くプログラムを処理します。
 - #elifは#if、#ifdef、#ifndefと対で使用します。
 - #elseは#ifと対で使用します。#elseと改行の間に字句列を記述してはいけません。ただし、コメントを記述することはできます。
 - #endifは#ifで制御する範囲の終了を示します。#ifコマンドを使用する場合は必ずこのコマンドを記述してください。
 - #if～#elif～#else～#endifの組み合わせはネストすることができます。ネストレベルは特に制限はありません(ただし、メモリ容量に依存します)。
 - 定数式の中にsizeof演算子、cast演算子、変数を使用することはできません。

#ifdef～#elif～#else～#endif

機能 条件コンパイルを行います(マクロが定義されているか否かを調べます)。

書式 #ifdef△識別子
:
#elif△定数式
:
#else
:
#endif

解説 ● #ifdefは、識別子が定義されている場合、後に続くプログラムを処理します。また以下のように記述することもできます。

```
#if△defined△識別子  
#if△defined△(識別子)
```

- #elseは#ifdefと対で使用します。#elseと改行の間に字句列を記述してはいけません。ただし、コメントを記述することはできます。
- #elifは#if、#ifdef、#ifndefと対で使用します。
- #endifは#ifdefで制御する範囲の終了を示します。#ifdefコマンドを使用する場合は必ずこのコマンドを記述してください。
- #ifdef～#else～#endifの組み合わせはネストすることができます。ネストレベルは特に制限はありません(ただし、メモリ容量に依存します)。

#ifndef～#elif～#else～#endif

機能 条件コンパイルを行います(マクロが定義されているか否かを調べます)。

書式 #ifndef△識別子
:
#elif△定数式
:
#else
:
#endif

解説 ● #ifndefは、識別子が定義されていない場合、後に続くプログラムを処理します。また以下のように記述することもできます。

```
#if△!defined△識別子  
#if△!defined△(識別子)
```

- #elseは#ifndefと対で使用します。#elseと改行の間に字句列を記述することはできません。ただし、コメントを記述することはできます。
- #elifは#if、#fdef、#ifndefと対で使用します。
- #endifは#ifndefで制御する範囲の終了を示します。#ifndefコマンドを使用する場合は必ずこのコマンドを記述してください。
- #ifndef～#else～#endifの組み合わせはネストすることができます。ネストレベルは特に制限はありません(ただし、メモリ容量に依存します)。

#include

- 機能** 指定したファイルの取り込みを行います。
- 書式**
- (1) #include△<ファイル名>
 - (2) #include△"ファイル名"
 - (3) #include△識別子
- 解説**
- (1) nc100の起動オプション-Iで指定されたディレクトリのファイルを取り込みます。ファイルが見つからない場合は、以下のディレクトリを検索します。
 - 環境変数INC100により設定された標準ディレクトリ
 - (2) カレントディレクトリのファイルを取り込みます。ファイルが見つからない場合は、以下のディレクトリを順番に検索します。
 - (1) 起動オプション-Iで指定されたディレクトリ
 - (2) 環境変数INC100により設定された標準ディレクトリ
 - (3) マクロ展開された識別子が、<ファイル名>又は"ファイル名"であるとき、そのファイルを(1)又は(2)の検索規則に準じてディレクトリから取り込みます。
 - ネストレベルは最大40です。
 - 該当するファイルが存在しないときはインクルードエラーとなります。

#line

- 機能** ファイル中の行番号を付け換えます。
- 書式** #line△整数△"ファイル名"
- 解説**
- ファイルの行番号及びファイル名を設定します。
 - ソースファイル名及び行番号を変更することができます。

#pragma

機能 NC100の拡張機能処理を指示します。

- 書式**
- (1) #pragma ROM△変数名
 - (2) #pragma SBDATA△変数名
 - (3) #pragma SB16DATA△変数名
 - (4) #pragma SECTION△既定セクションベース名 変更セクションベース名
 - (5) #pragma STRUCT△構造体のタグ名△unpack
 - (6) #pragma STRUCT△構造体のタグ名△arrange
 - (7) #pragma ADDRESS△変数名△絶対アドレス
 - (8) #pragma DMAC△変数名△DMACレジスタ名
 - (9) #pragma INTCALL△INT番号△アセンブラ関数名(レジスタ名, レジスタ名, ...)
 - (10) #pragma INTCALL△INT番号△C言語関数名()
 - (11) #pragma INTERRUPT△[/B /E /F /R /W]△割り込みベクタ番号△割り込み処理関数名
 - (12) #pragma PARAMETER△アセンブラ関数名(レジスタ名, レジスタ名, ..)
 - (13) #pragma ASM
 - (14) #pragma ENDASM
 - (15) #pragma JSRA△関数名
 - (16) #pragma JSRW△関数名
 - (17) #pragma PAGE
 - (18) #pragma __ASMMACRO△関数名(レジスタ名)
 - (19) #pragma MRCALL△S=スタックサイズ△INT番号 機能コードサービスコール名 (引数の型...)
 - (20) #pragma MRPARAMETER△サービスコール名 (引用の型...)
 - (21) #pragma ALMHANDLER△アラームハンドラ名
 - (22) #pragma CYCHANDLER△周期起動ハンドラ名
 - (23) #pragma INTHANDLER△[/E | /R]△割り込みハンドラ名
 - (24) #pragma TASK△タスクの開始関数名
 - (25) #pragma EXTMEM 変数名
 - (26) #pragma EXTMEM 関数名()

#pragma

- 解 説
- (1) romセクションへの配置機能
 - (2) SB相対アドレッシング使用変数記述機能
 - (3) SB相対16ビットディスプレイメントアドレッシング使用変数宣言機能
 - (4) セクションベース名変更機能
 - (5) 構造体配列制御機能
 - (6) 構造体配列制御機能
 - (7) 入出力変数の絶対アドレス指定機能
 - (8) 外部変数のDMACレジスタ割り付け機能
 - (9) ソフトウェア割り込み使用関数宣言機能
 - (10) ソフトウェア割り込み使用関数宣言機能
 - (11) 割り込み関数の記述機能
 - (12) レジスタ渡しのアセンブラ関数宣言機能
 - (13) インラインアセンブラ記述機能
 - (14) インラインアセンブラ記述機能
 - (15) 関数呼び出し命令指定機能
 - (16) 関数呼び出し命令指定機能
 - (17) 改ページ指定機能
 - (18) アセンブラマクロ関数宣言機能
 - (19) R32Cシリーズ用リアルタイムOSのサービスコールのインターフェース宣言機能
 - (20) R32Cシリーズ用リアルタイムOSのサービスコールのインターフェース宣言機能
 - (21) R32Cシリーズ用リアルタイムOSのアラームハンドラ関数を宣言機能
 - (22) R32Cシリーズ用リアルタイムOSの周期ハンドラ宣言機能
 - (23) R32C シリーズ用リアルタイムOS のカーネル管理割り込みハンドラ宣言機能
 - (24) R32Cシリーズ用リアルタイムOSのTASK開始関数宣言機能
 - (25) 0番地相対アドレッシング除外宣言機能
 - (26) 0番地相対アドレッシング除外宣言機能
- #pragmaでは上記25種類の処理機能のみを指定することができます。#pragmaに続けて上記以外の文字列、識別子を記述した場合、その処理指定は無視されます。
 - サポートしていない#pragmaを使用した場合、デフォルトでは警告を出力しません。nc100の起動オプション-Wunknown_pragma(-WUP)を指定したときのみ警告を出力します。

#undef

機能 マクロを未定義にします。

書式 #undef△識別子

解説

- マクロとして定義された識別子を無効にします。
- 以下の4つの識別子はコンパイラの予約語です。これらの識別子は常に有効にしておく必要がありますので、絶対に#undefで無効にしないでください。

__FILE__	ソースファイルの名前
__LINE__	現在のソースファイルの行番号
__DATE__	コンパイルの日付(形式:mm dd yyyy)
__TIME__	コンパイルの時間(形式:hh:mm:ss)

C.3.3 プリデファインドマクロ

NC100 では、予め以下のマクロが定義されています。

- R32C100
- NC100
- __INT_16__ (コンパイルオプション-fI16 (-fint_16) 使用時に定義されます)
- __CHAR_SIGNED__ (コンパイルオプション-fSC (-fsigned_char) 使用時に定義されます)

C.3.4 プリデファインドマクロの使用方法

【図C.14】のように、プリデファインドマクロはNC100以外のC言語プログラム中でマシン依存部を切り換えるとき等に使用します。

```
#ifdef NC100
#pragma ADDRESS port0 2H
#pragma ADDRESS port1 3H
#else
#pragma AD portA = 0x5F
#pragma AD portA = 0x60
#endif
```

図C.14 プリデファインドマクロの使用例

付録D C 言語実装仕様

本コンパイラが扱うデータの内部構造/配置、演算時等における符号拡張規則と、関数の呼び出し、および関数からの戻り値に関する規則を説明します。

D.1 データの内部表現

D.1.1 整数型

【表D.1】に整数型のデータが使用するバイト数を示します。

表D.1 整数型のデータサイズ

型	符号の有無	ビットサイズ	表現できる数値
_Bool	なし	8	0、1
char unsigned char	なし	8	0~255
signed char	有り	8	-128~127
int short signed int signed short	有り	16	-32768~32767
unsigned int unsigned short	なし	16	0~65535
int long signed int signed long	有り	32	-2147483648~2147483647
unsigned int unsigned long	なし	32	0~4294967295
long long signed long long	有り	64	-9223372036854775808~9223372036854775807
unsigned long long	なし	64	18446744073709551615
float	有り	32	1.17549435e-38F~3.40282347e+38F
double long double	有り	64	2.2250738585072014e-308~ 1.7976931348623157e+308
near ポインタ	なし	16	0~0xFFFF
far ポインタ	なし	32	0~0xFFFFFFFF

- _Bool型の符号指定はできません。
- char型は、符号指定がない場合、unsigned char型と解釈します。
- int型、short型は、符号指定がない場合signed int型、signed short型と解釈します。
- long型は、符号指定がない場合signed long型と解釈します。
- long long型は、符号指定がない場合signed long long型と解釈します。
- 構造体のビットフィールドメンバで符号指定がない型は、符号なしと解釈します。

- long long型のビットフィールドは使用できません。
- int型は32ビットで扱います。ただし、コンパイルオプション“-fint16(-fint_16)”を指定した場合は16ビットで扱います。

D.1.2 浮動小数点型

【表D.2】に浮動小数点型のデータが使用するバイト数を示します。

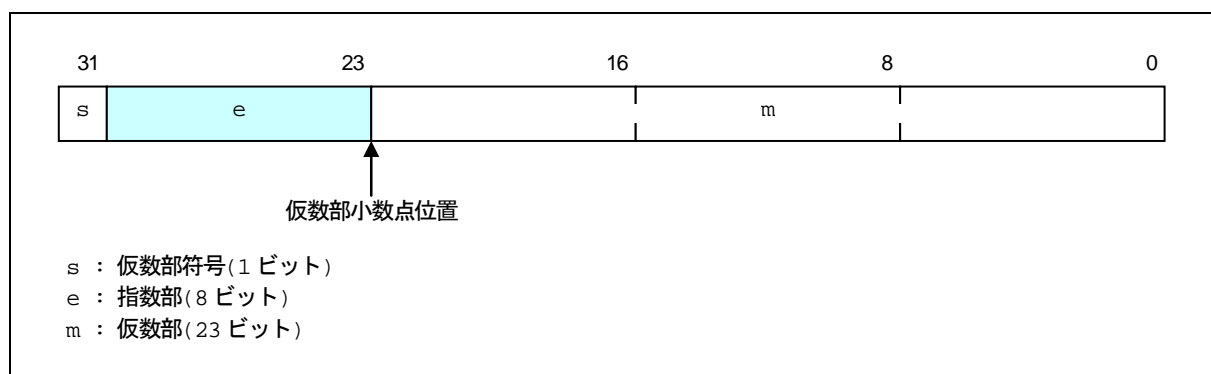
表D.2 浮動小数点型のデータサイズ

型	符号の有無	ビットサイズ	表現できる数値
float	有り	32	1.17549435e-38F ~ 3.40282347e+38F
double	有り	64	2.2250738585072014e-308 ~
long double			1.7976931348623157e+308

本コンパイラの浮動小数点フォーマットは、IEEE(The Institute of Electrical and Electronics Engineers)規格の形式に準拠しています。以下に、単精度/倍精度の浮動小数点フォーマットを示します。

(1) 単精度浮動小数点データフォーマット

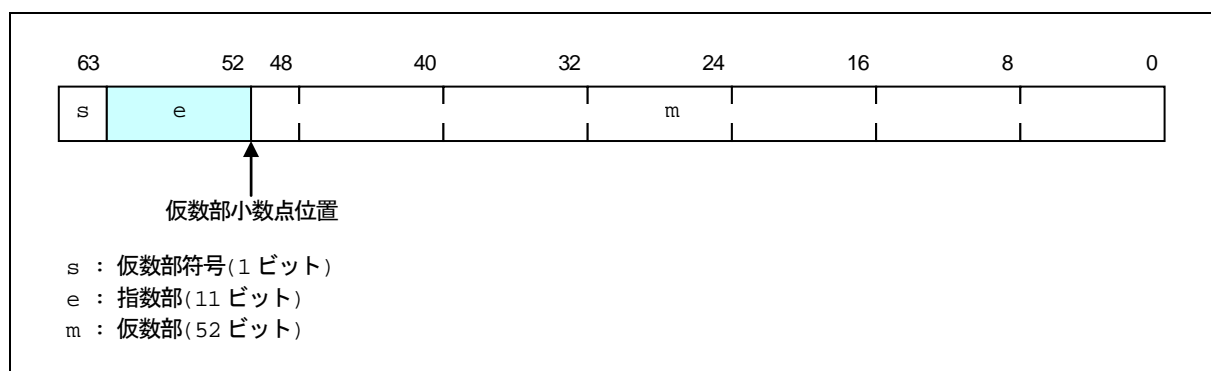
【図D.1】に示すデータ形式で2進数の浮動小数点(float)データを表現します。



図D.1 単精度浮動小数点データフォーマット

(2) 倍精度浮動小数点データフォーマット

【図D.2】に示すデータ形式で2進数の浮動小数点(double、long double)データを表現します。



図D.2 倍精度浮動小数点データフォーマット

D.1.3 列挙型

列挙型は、`unsigned int` 型と同じ内部表現となります。特に指定しない場合、メンバの出現順に 0、1、2 の整数値が与えられます。

D.1.4 ポインタ型

【表D.3】にポインタ型のデータが使用するバイト数を示します。

表D.3 ポインタ型のデータサイズ

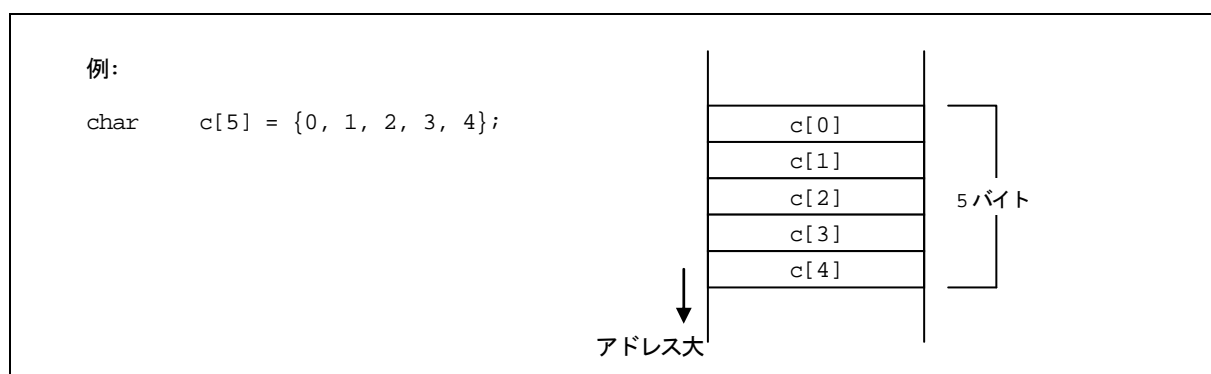
型	符号の有無	ビットサイズ	表現できる数値
ポインタ	なし	32	0~0xFFFFFFFF

ポインタはすべて `far` ポインタとして扱われます。そのため `near` ポインタで宣言されたポインタ変数に対して `far` ポインタとして扱われることを示すワーニング (`near pointer not supported, near qualifier ignored`) が出力されます。

なお、コンパイルオプション “`-WINP (-Wignore_near_pointer)`” を指定すると上記ワーニングは抑止されます。

D.1.5 配列型

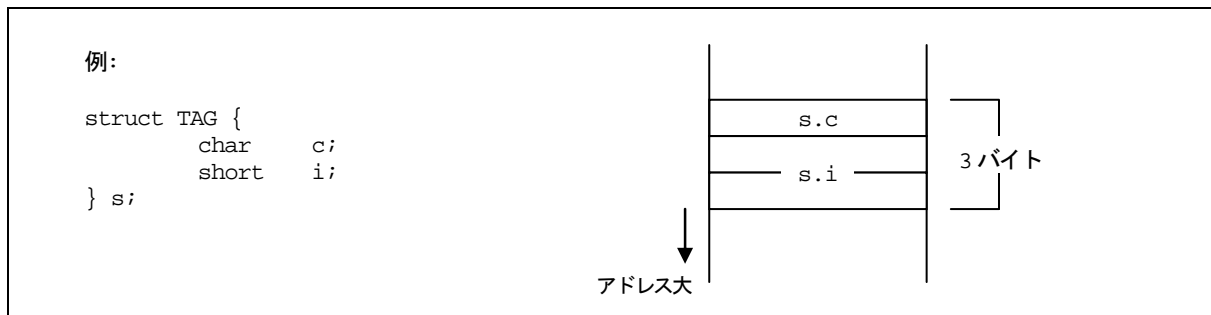
配列型は、要素のサイズ(バイト数)と要素数との積で表す領域に連続して配置されます(要素の出現順にメモリに配置されます)。【図D.3】に配置例を示します。



図D.3 配列の配置例

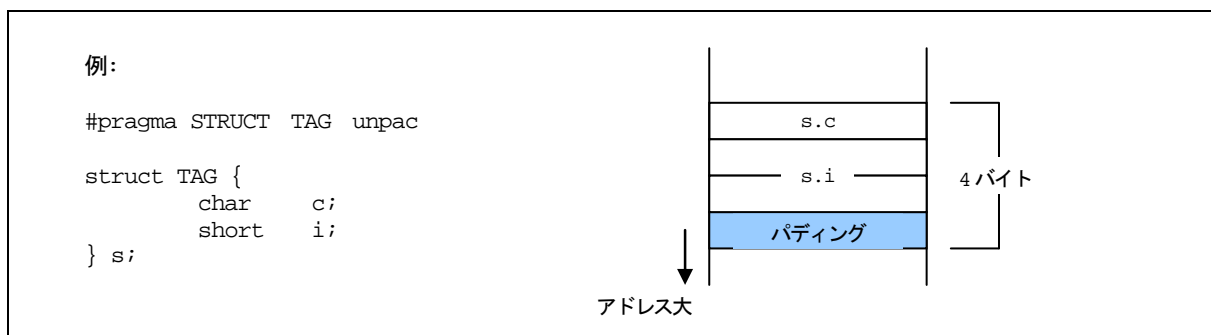
D.1.6 構造体型

構造体型は、メンバのデータを出現順に連続して配置します。【図D.4】に配置例を示します。



図D.4 構造体の配置例 (1)

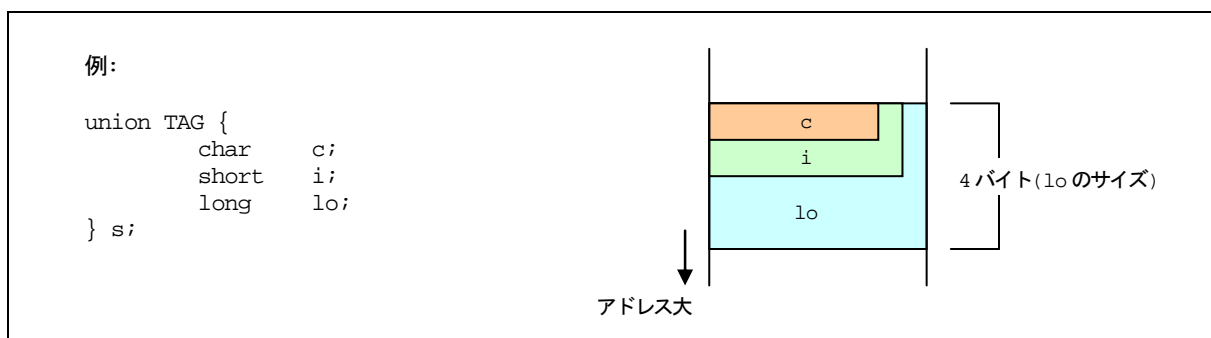
構造体は通常の場合、ワードアライメントを行いません。複数の構造体のメンバは連続して配置されます。ワードアライメントを行う場合は、拡張機能の`#pragma STRUCT`を使用します。`#pragma STRUCT`を使用することにより、メンバのサイズの合計が奇数バイトであるときに1バイトのパディングを付加します。【図D.5】に配置例を示します。



図D.5 構造体の配置例 (2)

D.1.7 共用体型

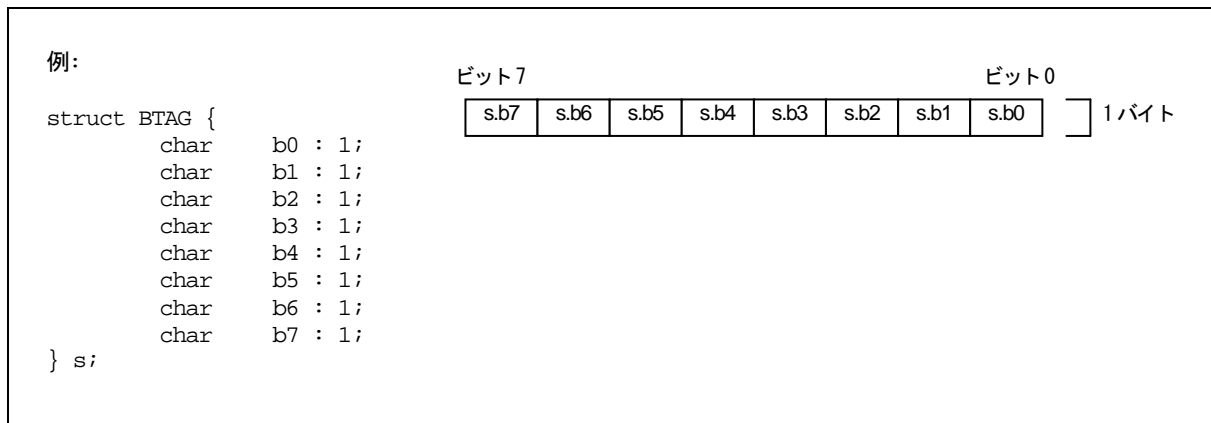
共用体型は、メンバの中で最大のデータサイズの領域をとります。【図D.6】に配置例を示します。



図D.6 共用体の配置例

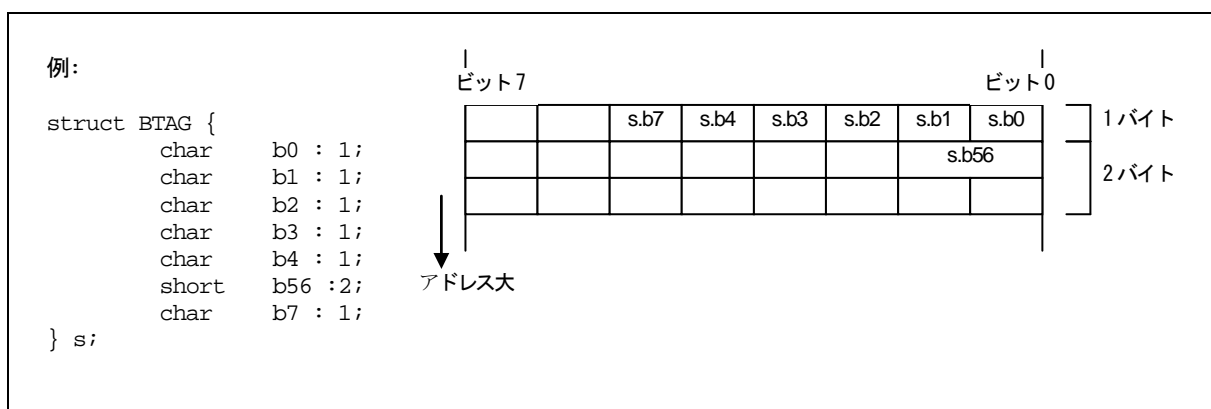
D.1.8 ビットフィールド型

ビットフィールド型は、最下位のビットから配置されます。【図D.7】に配置例を示します。



図D.7 ビットフィールドの配置例 (1)

ビットフィールドのメンバ中で、データ型が異なるものは次のアドレスに配置されます。この場合、同じデータ型のメンバは同じデータ型が配置されるアドレス上に最下位アドレスから連続して配置されます。



図D.8 ビットフィールドの配置例 (2)

● 注

- (1) ビットフィールドのメンバの型は、符号指定が無い場合 `unsigned` 型とみなします。
- (2) `long long` 型のビットフィールドは宣言できません。

D.2 符号拡張規則

ANSI規格等で定められた標準のC言語仕様では、char型のデータは演算時等においてint型に符号拡張して処理を行う規則を記しています。この仕様は、【図D.9】に示すようなchar型の演算を行うときに、演算の途中において、char型で表現できる最大値をオーバーフローして結果が予期しない値になることを防ぐためです。

```
void    func(void)
{
    char    c1, c2, c3;

    c1 = c2 * 2 / c3;
}
```

図D.9 C言語のサンプルプログラム例

本コンパイラでは、デフォルトでコード効率と実行速度を重視したコードを生成するために、char型をint型に符号拡張しません。この仕様は、コンパイルオプション"-fansi"又は"-fextend_to_int(-fETI)"を使用することにより無効となり、標準のC言語と同様の符号拡張を行います。

コンパイルオプション"-fansi"又は"-fextend_to_int(-fETI)"を使用せず、【図D.9】のように演算結果をchar型に代入するような演算を記述する場合は、char型で表現できる最小値及び最大値¹が演算途中でオーバーフローしないように注意してください。

¹本コンパイラでは、char型で表現できる値の範囲は以下のとおりです。

unsigned char型 0~255

signed char型 -128~127

D.3 関数呼び出し規則

D.3.1 戻り値に関する規則

関数から戻り値を返す場合、戻り値の型が整数型、ポインタ型、浮動小数点型の場合は、レジスタ渡しになります。
【表D.4】に戻り値に関する呼び出し規則を示します。

表D.4 戻り値に関する呼び出し規則

戻り値の型	規則
char 型 _Bool 型	R0L レジスタ
int 型 (16 ビット) short int 型	R0 レジスタ
int 型 (32 ビット) float 型 long 型	R2R0 レジスタ
ポインタ型	A0 レジスタ
long long 型 double 型 long double 型	A1A0 レジスタ (上位 32 ビットは A1 レジスタに、下位 32 ビットは A0 レジスタに格納)
構造体型 共用体型	呼び出しを行う直前に、戻り値を格納するための領域を指す far アドレスをスタックに積みま ず。呼び出された関数はリターンする前にスタックに積まれた far アドレスで指す領域に戻り 値を書き込みます。

D.3.2 引き数渡しに関する規則

本コンパイラは、関数への引数渡しの方法として、レジスタ渡しとスタック渡しの 2 通りがあります。

(1) 引き数のレジスタ渡し

以下に示す条件を満たす場合、【表D.5】、【表D.6】、【表D.7】中の「使用するレジスタ」を用いて引き数を渡しま
す。

- 関数のプロトタイプ宣言²を行ない、関数呼び出し時に引数の型が確定している。
- プロトタイプ宣言に可変引数"..."を使用していない。
- 関数の引数の型として、【表D.5】、【表D.6】、【表D.7】の引数と引数の型が一致している。

²本コンパイラでは、プロトタイプ宣言を行なった時のみ、レジスタ渡しを適応します。K&R形式の記述を行なった場合は、すべての引数をスタック渡し
で行ないます。

また、C言語の言語仕様上、関数に対してプロトタイプ宣言を行なう記述形式とK&R形式の記述を混在すると、引数が関数に正しく渡されない場合があ
ることに注意してください。

上記理由により、プロトタイプ宣言を行なう記述形式に統一してC言語ソースファイルを記述することを推奨しています。

表D.5 レジスタ引数渡し規則 (NC100)

引数	引数の型	使用するレジスタ
第1引数	_Bool型、char型	R0L
	int型(16ビット)、short型	R0
	int型(32ビット) float型、long型	R2R0
	ポインタ型	A0
	long long型、double型 long double型	A1A0
第2引数	_Bool型、char型	R1L
	int型(16ビット)、short型	R1
	int型(32ビット) float型、long型	R3R1
	ポインタ型	A2
	long long型、double型 long double型	A3A2
第3引数	int型(16ビット)、short型	R4
	int型(32ビット) float型、long型	R6R4
	ポインタ型	R6R4
第4引数	int型(16ビット)、short型	R5
	int型(32ビット) float型、long型	R7R5
	ポインタ型	R7R5

表D.6 レジスタ引数渡し規則 (NC308)

引数	引数の型	使用するレジスタ
第1引数	_Bool型 char型	R0Lレジスタ
	int型 nearポインタ型	R0レジスタ

表D.7 レジスタ引数渡し規則 (NC30)

引数	引数の型	使用するレジスタ
第1引数	_Bool型 char型	R1Lレジスタ
	int型 nearポインタ型	R1レジスタ
	int型 nearポインタ型	R2レジスタ

(2) 引数のスタック渡し

レジスタ渡しの条件を満たさない引数は、すべてスタック渡しになります。引き数の渡し方をまとめると、【表D.8】、【表D.9】、【表D.10】の様になります。

表D.8 関数の引き数渡しの規則 (NC100)

引数の型	第1引数	第2引数	第3引数	第4引数	第5引数以降
_Bool型、char型	R0L	R1L	スタック	スタック	スタック
int型 (16ビット) short型	R0	R1	R4	R5	スタック
int型 (32ビット) float型、long型	R2R0	R3R1	R6R4	R7R5	スタック
ポインタ型	A0	A2	R6R4	R7R5	スタック
long long型、double型 long double型	A1A0	A3A2	スタック	スタック	スタック

表D.9 関数の引き数渡しの規則 (NC308)

引数の型	第1引数	第2引数	第3引数以降
_Bool型 char型	R0L レジスタ	スタック	スタック
int型 near ポインタ型	R0 レジス	スタック	スタック

表D.10 関数の引き数渡しの規則 (NC30)

引数の型	第1引数	第2引数	第3引数以降
_Bool型 char型	R1L レジスタ	R2 レジスタ	スタック
int型 near ポインタ型	R1 レジスタ	スタック	スタック

D.3.3 関数のアセンブリ言語シンボルへの変換規則

C 言語ソースファイルでの関数定義時の関数名は、アセンブラソースファイルでの関数の先頭ラベルとして使用します。

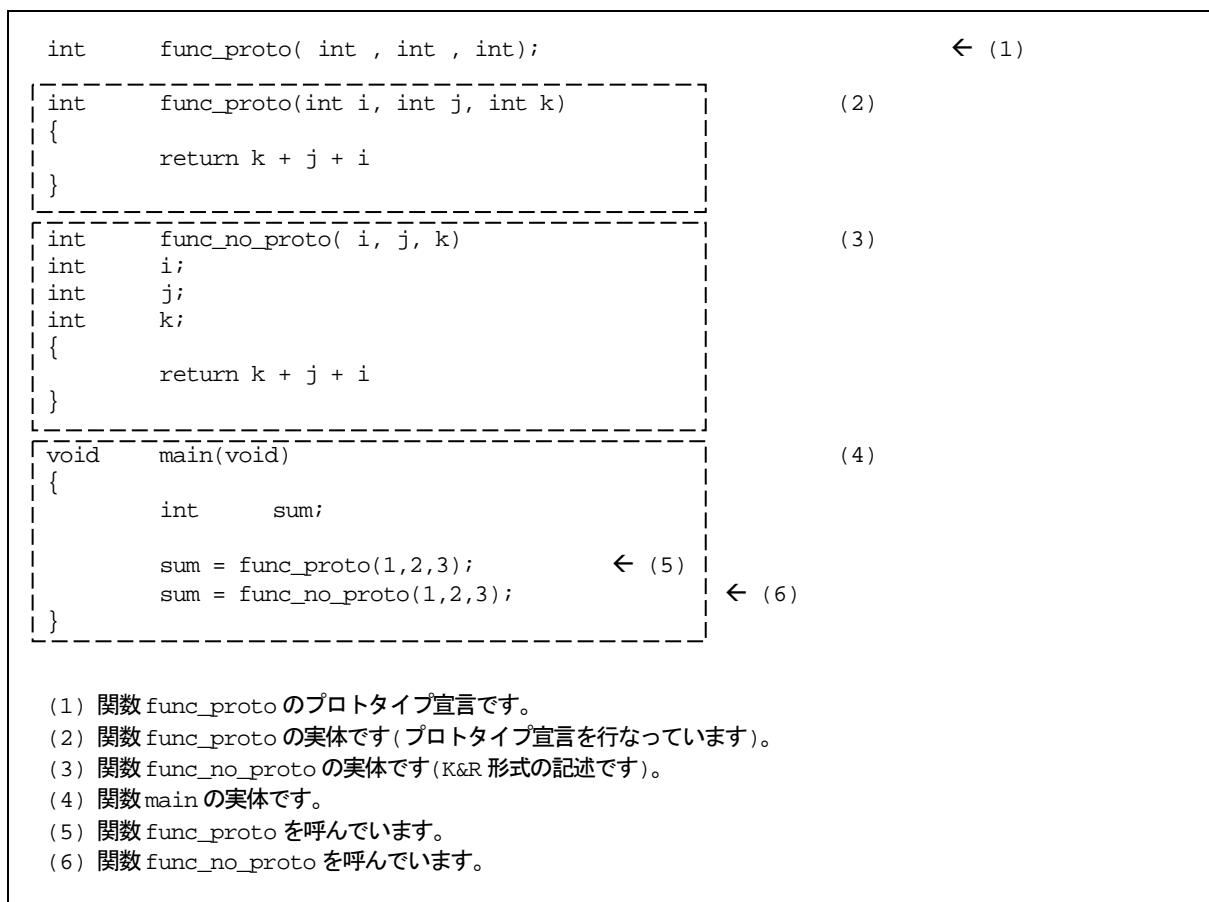
アセンブラソースファイルでの関数の先頭ラベルは、C言語ソースファイルでの関数名の先頭に_(アンダーバー)あるいは\$(ダラー)を付加したもの、または、関数名それ自身になります。付加文字列と文字列が付加される条件を【表D.11】に示します。

表D.11 関数に文字列の付加される条件

付加文字列	条件
\$(ダラー)	一つでも引数がレジスタ渡しとなる関数
_(アンダーバー)	上記条件以外の関数 ³

【図D.10】に示すプログラムは、関数の引数がレジスタ引数を持つものと、関数の引数をスタック渡しのみで扱う例です。

³ #pragma INTCALLで指定した関数は関数名を出力しません。



図D.10 関数呼び出しのサンプルプログラム (sample.c)

上記サンプルプログラムのコンパイル結果について、関数func_protoの定義(2)の部分を【図D.11】に、関数func_no_protoの定義(3)の部分を【図D.12】に、関数func_protoと関数func_no_protoの呼び出し(4)の部分を【図D.13】に示します。

```

;## #   FUNCTION func_proto
;## #   REGISTER ARG   (    i) size  4, REGISTER R2R0   ← (9)
;## #   REGISTER ARG   (    j) size  4, REGISTER R3R1   ← (8)
;## #   REGISTER ARG   (    k) size  4, REGISTER R6R4   ← (7)
;## #   ARG Size(0)      Auto Size(0)   Context Size(4)

        .SECTION program, CODE, ALIGN
        ._file   'test.c'
        .align
        ._line   4
;## #   C_SRC :      {
        .glb     $func_proto
$func_proto:                                ← (10)
        ._line   5
;## #   C_SRC :      return k + j + i;
        add.l    R3R1,R6R4    ; j
        add.l    R6R4,R2R0    rts
E1:

```

- (7) 第3引数kをレジスタ渡しにしています。
(8) 第2引数jをレジスタ渡しにしています。
(9) 第1引数iをレジスタ渡しにしています。
(10) 関数 func_proto の先頭アドレスです。

図D.11 サンプルプログラム(sample.c)のコンパイル結果 (1)

【図D.10】のサンプルプログラム(sample.c)のコンパイル結果(1)では、関数func_protoは、プロトタイプ宣言を行なっているため第1、第2、第3引数はレジスタ渡しになります。

また、関数の引数がレジスタ渡しになるため、関数の先頭アドレスのシンボル名は、C 言語ソースファイルに記述した" func_proto" の前に\$(ダラー) を付加した"\$func_proto"になります。

```

;## #   FUNCTION func_no_proto
;## #   FRAME  ARG (    i) size  4,  offset 8
;## #   FRAME  ARG (    j) size  4,  offset 12
;## #   FRAME  ARG (    k) size  4,  offset 16
;## #   ARG Size(12)      Auto Size(0)   Context Size(8)

        .align
        ._line   12
;## #   C_SRC :      {
        .glb     _func_no_proto
_func_no_proto:                                ← (12)
        enter   #00H
        ._line   13
;## #   C_SRC :      return k + j + i;
        mov.l   16[FB],R2R0    ; k
        add.l   12[FB],R2R0    ; j
        add.l   8[FB],R2R0     ; i
        exitd
E2:

```

- (11) すべての引き数をスタック渡しにしている。
(12) 関数 func_no_proto の先頭アドレスです。

図D.12 サンプルプログラム(sample.c)のコンパイル結果 (2)

【図D.10】のサンプルプログラム(sample.c)のコンパイル結果(2)では、関数func_no_protoはK&R形式の記述を行なっているため、すべての引数がスタック渡しになります。

また、関数の引数にレジスタ渡しを含まないため、関数の先頭アドレスのシンボル名は、C 言語ソースファイルに記述した"func_no_proto"の前に_(アンダーバー)を付加した"_func_no_proto"になります。

```

### #   FUNCTION main
### #   FRAME   AUTO   (   sum)           size 4, offset -4
### #   ARG Size(4)   Auto Size(4)       Context Size(8)

        .align
        .line 17
### # C_SRC :   {
        .glb   _main
_main:
        enter  #04H
        .line 20
### # C_SRC :   sum = func_proto(1,2,3);
        mov.l  #00000003H,R6R4
        mov.l  #00000002H,R3R1
        mov.l  #00000001H,R2R0
        jsr   $func_proto
        mov.l  R2R0,-4[FB]      ; sum
        .line 21
### # C_SRC :   sum = func_no_proto(1,2,3);
        push.l #00000003H
        push.l #00000002H
        push.l #00000001H
        jsr   _func_no_proto
        add.l #0cH,SP
        mov.l R2R0,-4[FB]      ; sum
        .line 22
### # C_SRC :   }
        exitd
E3:

        .END

```

図D.13 サンプルプログラム(sample.c)のコンパイル結果 (3)

【図D.13】において、(13) の部分はfunc_protoの呼び出しを、(14) の部分はfunc_no_protoの呼び出しを行っています。

D.3.4 関数間のインターフェース

【図D.14】に示すプログラムにおいて、スタックフレームの構築及び解放の処理を【図D.17】と【図D.18】に示します。なお、【図D.15】と【図D.16】は、【図D.14】のプログラムをコンパイルした結果、出力されたアセンブリ言語プログラムです。

```
iint func( int, int, int);

void    main(void)
{
    int ans;
    int i = 0x1111; ← func への引数
    int j = 0x2222; ← func への引数
    int k = 0x3333; ← func への引数
    ans = func( i, j ,k);
}

int func( int x, int y, int z )
{
    int sum;
    int s = 0x4444;
    int t = 0x5555;
    int u = 0x6666;
    sum = s + t + u + x + y + z ;
    return sum;          ← main への戻り値
}
```

図D.14 C言語サンプルプログラム

```

;## #   FUNCTION main
;## #   FRAME   AUTO   (   ans)       size 4, offset -4
;## #   ARG Size(4)   Auto Size(4)   Context Size(8)

        .SECTION program, CODE, ALIGN
        ._file   'interface.c'
        .align
        ._line   4
;## # C_SRC :   {
        .glb     _main
_main:
        enter   #04H                ← (1)
        ._line   6                    ← (2)
;## # C_SRC :   int i = 0x1111;
        mov.l   #00001111H,R2R0   ; i
        ._line   7
;## # C_SRC :   int j = 0x2222;
        mov.l   #00002222H,R3R1   ; j
        ._line   8
;## # C_SRC :   int k = 0x3333;
        mov.l   #00003333H,R6R4   ; k
        ._line   9
;## # C_SRC :   ans = func( i, j ,k);
        jsr    $func                ← (3)
        mov.l   R2R0,-4[FB]         ; ans ← (7)
        ._line  10
;## # C_SRC :   }
        exitd
E1:

```

図D.15 アセンブリ言語サンプルプログラム (1)

```

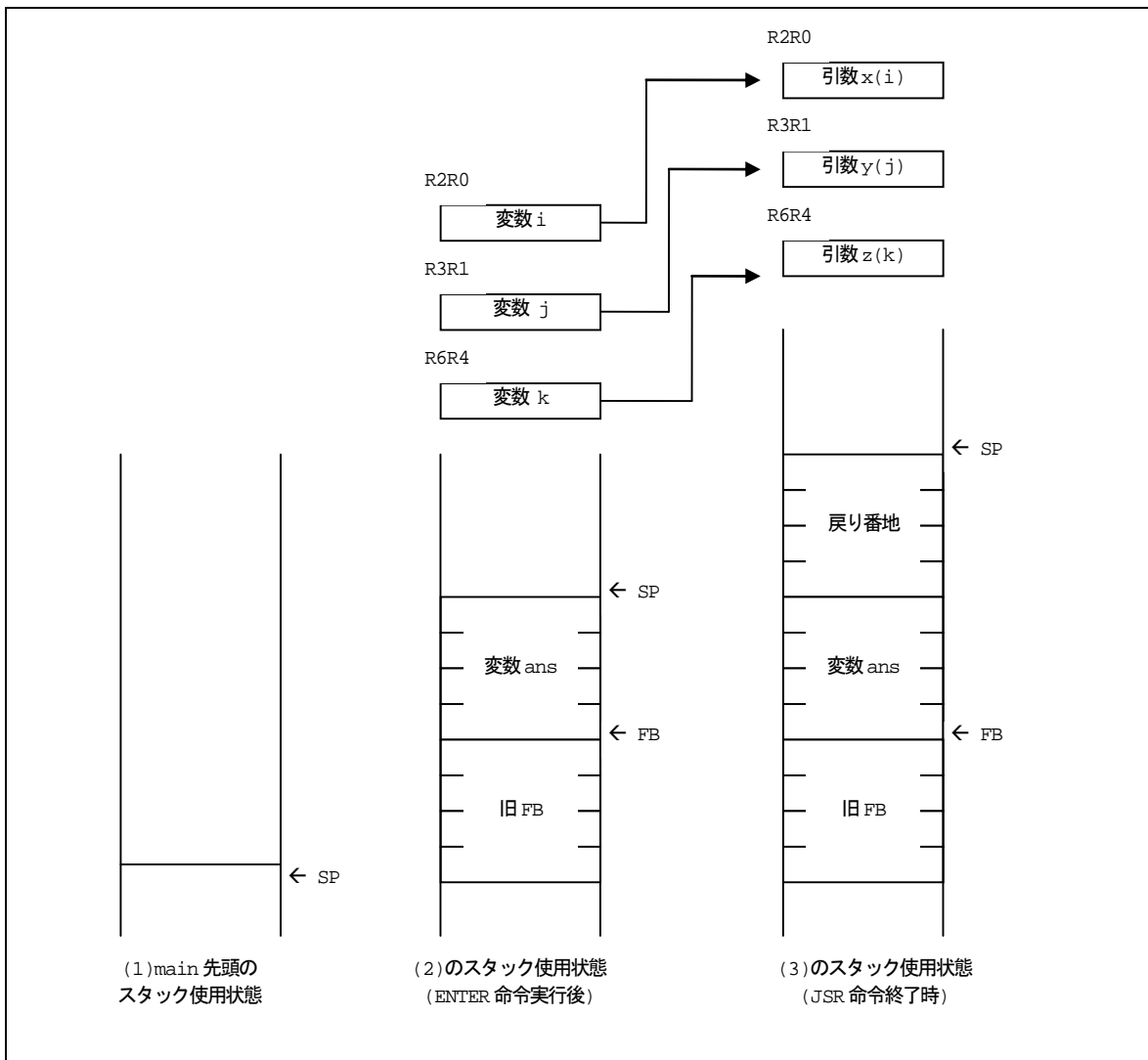
;## #   FUNCTION func
;## #   FRAME   AUTO   (   u)           size 4, offset -4
;## # REGISTER ARG (   x) size 4, REGISTER R2R0
;## # REGISTER ARG (   y) size 4, REGISTER R3R1
;## # REGISTER ARG (   z) size 4, REGISTER R6R4
;## #   ARG Size(0)      Auto Size(4)    Context Size(8)

        .align
        ._line 13
;## # C_SRC :   {
        .glob $func
$func:
        enter   #04H                ← (4)
        ._line 15
;## # C_SRC :   int s = 0x4444;
        mov.l   #00004444H,R7R5 ; s
        ._line 16
;## # C_SRC :   int t = 0x5555;
        mov.l   #00005555H,A0   ; t
        ._line 17
;## # C_SRC :   int u = 0x6666;
        mov.l   #00006666H,-4[FB]; u
        ._line 18
;## # C_SRC :   sum = s + t + u + x + y + z ;
        add.l   A0,R7R5 ; t
        add.l   -4[FB],R7R5 ; u
        add.l   R7R5,R2R0
        add.l   R2R0,R3R1
        mov.l   R6R4,R2R0 ; z z
        add.l   R3R1,R2R0 ; sum ← (5)
        ._line 19
;## # C_SRC :   return sum;
        exitd   ← (6)
E2:

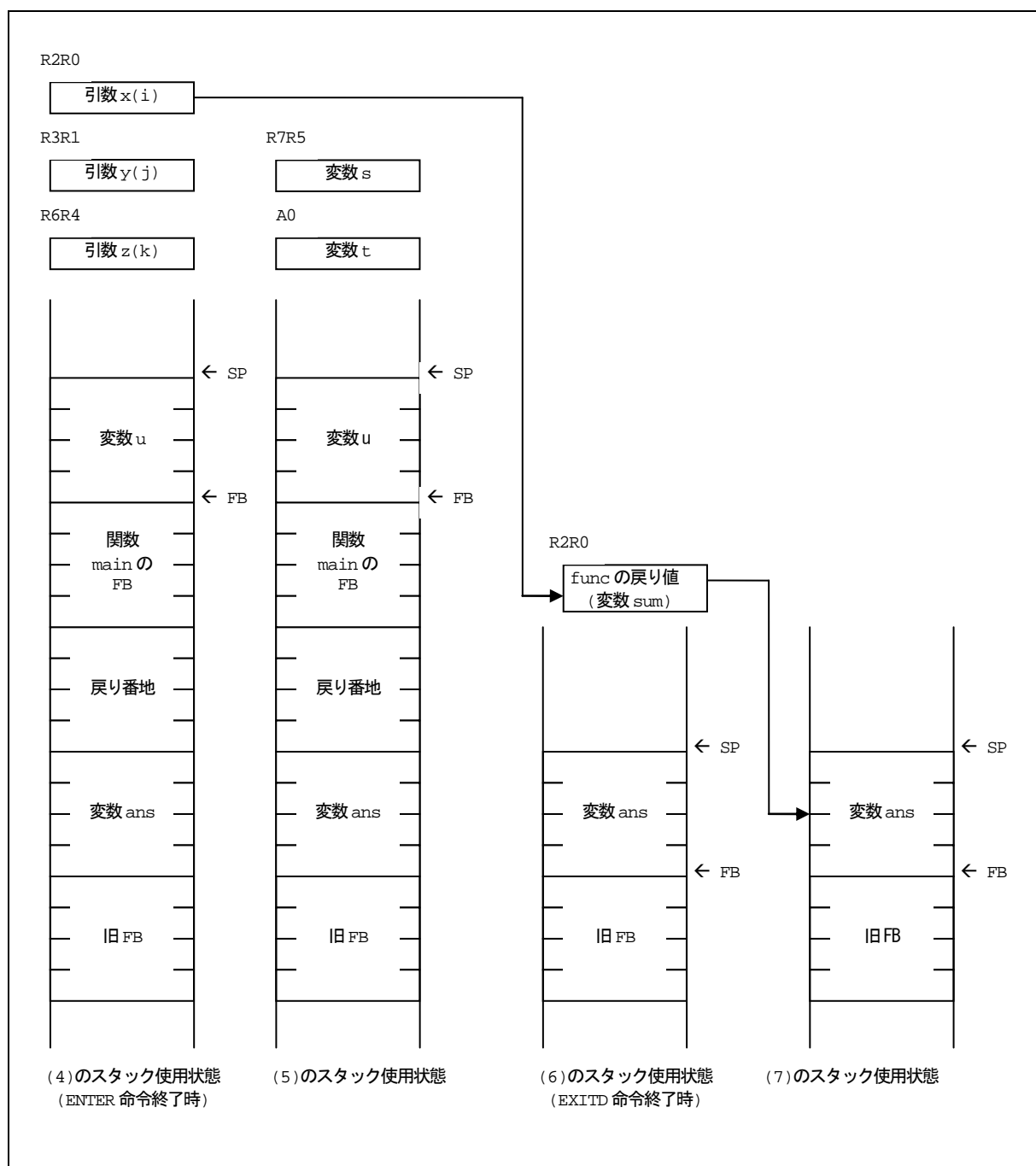
```

図D.16 アセンブリ言語サンプルプログラム (2)

【図D.15】中における (1) → (2) → (3) の処理（関数mainの入り口処理および関数funcの呼び出し）を【図D.17】に、(4) → (5) → (6) → (7) の処理（関数funcで使用するスタックフレームの構築処理および関数funcから関数mainへの戻り処理）を【図D.18】に、各々のスタック及びレジスタの遷移を示します。



図D.17 関数mainの入り口処理および関数funcの呼び出し



図D.18 関数funcで使用するスタックフレームの構築処理および関数funcから関数mainへの戻り処理

D.4 auto変数の領域確保

記憶クラスautoの変数は、マイコンのスタック上に配置されます。【図D.19】に示すようなC言語ソースプログラムでは、記憶クラスautoの変数が有効となる領域が互いに重ならない場合、1つの領域のみ確保を行い、複数の変数でその領域を共有します。

```

void    func(void)
{
    int    i, j, k;

    for( i=0 ; i<=0 ; i++ ){
        処理                               iの有効範囲
    }
    :
    (省略)
    :
    for( j=0xFF ; j<=0 ; j-- ){
        処理                               jの有効範囲
    }
    :
    (省略)
    :
    for( k=0 ; k<=0 ; k++ ){
        処理                               kの有効範囲
    }
}

```

図D.19 C言語ソースプログラム例

この例では、3つのauto変数i、j、kは有効となる範囲が重ならないため、同じ2バイトの領域(FBからのオフセット位置)を共有します。【図D.19】をコンパイルして生成されたアセンブリ言語ソースファイルを【図D.20】に示します。

```

;### FUNCTION    func
;###   FRAME AUTO      (      k)    size 4, offset -4      ← (1)
;###   FRAME AUTO      (      j)    size 4, offset -4      ← (2)
;###   FRAME AUTO      (      i)    size 4, offset -4      ← (3)
.section program
._file  'auto.c'
._line  3
.glb    _func
_func:
enter   #04H
:
(以下省略)

```

※3つの auto 変数は(1)~(3)に示すように、FB オフセット-2 の領域を共有しています。

図D.20 アセンブリ言語ソースプログラム例

D.5 レジスタの退避

C 言語の関数を呼び出す場合のレジスタの退避規則を以下に示します。

(a) 関数の呼び出し側で退避するレジスタ

- 呼び出し側で使用するレジスタ。

(b) 呼び出された関数の入口処理で退避するレジスタ

- 退避しません

付録E 標準ライブラリ

E.1 標準ヘッダファイル

標準ライブラリを使用する場合、その関数の定義を行っているヘッダファイルをインクルードする必要があります。標準ヘッダファイルの機能との仕様の詳細を説明します。

E.1.1 標準ヘッダファイルの概要

本コンパイラは、【表E.1】に示すように 15 個の標準ヘッダファイルを用意しています。

表E.1 標準ヘッダファイル一覧表

ヘッダファイル名	内容
assert.h	プログラムの診断情報の出力
ctype.h	文字判定関数のマクロ宣言
errno.h	エラー番号の定義
float.h	浮動小数点数の内部表現に関する各種制限値の定義
limits.h	コンパイラの内部処理に関する各種制限値の定義
locale.h	関数/マクロの地域化
math.h	数値計算
setjmp.h	分岐関数、分岐関数で使用する構造体の定義
signal.h	非同期割り込みを処理するための定義/宣言
stdarg.h	可変個の実引数を持つ関数の宣言と定義
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の定義
stdio.h	(1) FILE 構造体の定義 (2) ストリーム名の定義 (3) 入出力関数のプロトタイプ宣言
stdlib.h	メモリ管理関数、終了関数のプロトタイプ宣言
string.h	文字列操作関数、メモリ操作関数のプロトタイプ宣言
time.h	現在の暦時間を得る

E.1.2 標準ヘッダファイルリファレンス

本コンパイラが用意している標準ヘッダファイルの詳細仕様を説明します。ヘッダファイルは、アルファベット順に掲載しています。

ヘッダファイルの内部で宣言している本コンパイラの標準関数と、データ型の数値表現における制限値を定義しているマクロを、対応するヘッダファイルと共に説明します。

assert.h

機能 関数assertを定義しています。

ctype.h

機能 文字操作関数を宣言、及びマクロを定義しています。文字操作関数を以下に示します。

関数名	機能
isalnum	英数字の判定
isalpha	英字の判定
iscntrl	コントロール文字の判定
isdigit	数字の判定
isgraph	英数字、ブランク以外の文字判定
islower	英数字、ブランク以外の文字判定
isprint	ブランク文字を含む印字可能文字の判定
ispunct	区切り文字の判定
isspace	ブランク、タブ、改行の判定
isupper	英大文字の判定
isxdigit	16進数字の判定
tolower	大文字から小文字への変換
toupper	小文字から大文字への変換

errno.h

機能 エラー番号を定義しています。

float.h

機能

浮動小数点数の内部表現に関する各種制限値を定義しています。以下に、浮動小数点数の制限値を定義したマクロを示します。

本コンパイラでは、long doubleはdoubleとして扱います。long doubleの各制限値はdoubleと同じに定義しています。

マクロ名	内容	定義値
DBL_DIG	double 型の 10 進精度の最大桁数	15
DBL_EPSILON	1.0+DBL_EPSILON が 1.0 と異なると判断できる正の最小値	2.2204460492503131e-16
DBL_MANT_DIG	double 型の浮動小数点数値を基数に合わせて表現したときの仮数部の最大桁数	53
DBL_MAX	double 型の変数が値として持つことができる最大値	1.7976931348623157e+308
DBL_MAX_10_EXP	double 型の浮動小数点数値として表現できる 10 のべき乗の最大値	308
DBL_MAX_EXP	double 型の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	1024
DBL_MIN	double 型の変数が値として持つことができる最小値	2.2250738585072014e-308
DBL_MIN_10_EXP	double 型の浮動小数点数値として表現できる 10 のべき乗の最小値	-307
DBL_MIN_EXP	double 型の浮動小数点数値として表現できる基数のべき乗の最小値	-1021
FLT_DIG	float 型の 10 進精度の最大桁数	6
FLT_EPSILON	1.0+FLT_EPSILON が 1.0 と異なると判断できる正の最小値	1.19209290e-07F
FLT_MANT_DIG	float 型の浮動小数点数値を基数に合わせて表現したときの仮数部の最大桁数	24
FLT_MAX	float 型の変数が値として持つことができる最大値	3.40282347e+38F
FLT_MAX_10_EXP	float 型の浮動小数点数値として表現できる 10 のべき乗の最大値	38
FLT_MAX_EXP	float 型の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	128
FLT_MIN	float 型の変数が値として持つことができる最小値	1.17549435e-38F

float.h

機能	マクロ名	内容	定義値
	FLT_MIN_10_EXP	float 型の浮動小数点数値として表現できる 10 のべき剰の最小値	-37
	FLT_MIN_EXP	float 型の浮動小数点数値として表現できる 基数のべき剰の最小値	-125
	FLT_RADIX	浮動小数の指数の基数	2
	FLT_ROUNDS	浮動小数点数のまるめ方法	1(四捨五入)

limits.h

機能

コンパイラの内部処理に関する各種制限値を定義しています。以下に、各種制限値を定義したマクロを示します。

マクロ名	内容	定義値
MB_LEN_MAX	マルチバイト文字型のバイト数の最大値	1
CHAR_BIT	char 型のビット数	8
CHAR_MAX	char 型の変数が値として持つことができる最大値	255 または 127(コンパイルオプション-fsigned_char 指定時)
CHAR_MIN	char 型の変数が値として持つことができる最小値	0 または -128(コンパイルオプション-fsigned_char 指定時)
SCHAR_MAX	signed char 型の変数が値として持つことができる最大値	127
SCHAR_MIN	signed char 型の変数が値として持つことができる最小値	-128
INT_MAX	int 型の変数が値として持つことができる最大値	32767 または 2147483647 (コンパイルオプション-fl16(-fint_16)指定時)
INT_MIN	int 型の変数が値として持つことができる最小値	-32768 または -2147483648 (コンパイルオプション-fl16(-fint_16)指定時)
SHRT_MAX	short int 型の変数が値として持つことができる最大値	32767
SHRT_MIN	short int 型の変数が値として持つことができる最小値	-32768
LONG_MAX	long 型の変数が値として持つことができる最大値	2147483647
LONG_MIN	long 型の変数が値として持つことができる最小値	-2147483648
LLONG_MAX	signed long long int 型の変数が値として持つことができる最大値	9223372036854775807
LLONG_MIN	signed long long int 型の変数が値として持つことができる最小値	-9223372036854775808
UCHAR_MAX	unsigned char 型の変数が値として持つことができる最大値	255

limits.h

機能

マクロ名	内容	定義値
UINT_MAX	unsigned int 型の変数が値として持つことができる最大値	65535
USHRT_MAX	unsigned short int 型の変数が値として持つことができる最大値	65535
ULONG_MAX	unsigned long int 型の変数が値として持つことができる最大値	4294967295
ULLONG_MAX	unsigned long long int 型の変数が値として持つことができる最大値	18446744073709551615

locale.h

機能

プログラムの地域化を操作するマクロと関数を定義/宣言しています。プロトタイプを宣言している関数を以下に示します。

関数名	機能
localeconv	構造体 lconv を初期化
setlocale	プログラムのロケール情報の設定と検索

math.h

機能

数学関数のプロトタイプを宣言しています。プロトタイプを宣言している関数を以下に示します。

関数名	機能
acos	逆コサインを計算
asin	逆サインを計算
atan	逆タンジェントを計算
atan2	逆タンジェントを計算
ceil	整数繰り上げ値を計算
cos	コサインを計算
cosh	双曲線コサインを計算
exp	指数関数を計算
fabs	倍精度浮動小数の絶対値を計算
floor	整数繰り下げ値を計算
fmod	剰余計算
frexp	浮動小数を仮数部と指数部に分割
labs	long 型整数の絶対値を計算
ldexp	浮動小数の巾を計算
log	自然対数を計算
log10	常用対数を計算
modf	実数を仮数部と指数部に分割
pow	巾乗計算
sin	サインを計算
sinh	双曲線サインを計算
sqrt	数値の平方根を計算
tan	タンジェントを計算
tanh	双曲線タンジェントを計算

setjmp.h

機能 分岐関数のプロトタイプ宣言と、その関数で使用する構造体を定義しています。プロトタイプを宣言している関数を以下に示します。

関数名	機能
longjmp	大域ジャンプ
setjmp	大域ジャンプのためのスタック環境の設定

signal.h

機能 非同期割り込みを処理するためのマクロと関数を定義/宣言しています。

stdarg.h

機能 可変長の引数リストを処理するためのルーチンを定義しています。

stddef.h

機能 各標準ヘッダファイルで共通に使用するマクロ名を定義しています。

stdio.h

機能 FILE構造体/ストリーム名の定義と、入出力関数のプロトタイプを宣言しています。プロトタイプを宣言している関数を以下に示します。

種別	関数名	機能
初期化	init	マイコンの入出力の初期化
	clearerr	エラー状態指示子を初期化(クリア)
入力	fgetc	一文字入力
	getc	一文字入力
	getchar	stdin からの一文字入力
	fgets	一行入力
	gets	stdin からの一行入力
	fread	指定データ数入力
	scanf	stdin からの書式付き入力
	fscanf	書式付き入力
	sscanf	文字列からの書式付きデータ入力
出力	fputc	一文字出力
	putc	一文字出力
	putchar	stdout への一文字出力
	fputs	一行出力
	puts	stdout への一行出力
	fwrite	指定データ数出力
	perror	stdout へのエラーメッセージ出力
	printf	stdout への書式付き出力
	fflush	出力バッファのストリームをフラッシュ
	Fprintf	書式付き出力
	sprintf	書式付き文字列設定
	vfprintf	ストリームへの書式付き出力
	vprintf	stdout への書式付き出力
	vsprintf	バッファへのへの書式付き出力
返還	ungetc	一文字入力の返還
判定	ferror	入出力エラーの判定
	feof	EOF(End Of File)の判定

stdlib.h

機能: メモリ管理関数、終了関数のプロトタイプを宣言しています。プロトタイプを宣言している関数を以下に示します。

関数名	機能
abort	プログラムの実行を終了
abs	int 型整数の絶対値を計算
atof	文字列を double 型浮動小数に変換
atoi	文字列を int 型に変換
atol	文字列を long 型に変換
bsearch	配列内のバイナリサーチを行う
calloc	メモリの確保と 0(ゼロ)による初期化
div	int 型整数の除算と剰余
free	メモリの解放
labs	long 型整数の絶対値を計算
ldiv	long 型整数の除算と剰余
malloc	メモリの確保
mblen	マルチバイト文字列の長さを計算
mbstowcs	マルチバイト文字列をワイド文字列に変換
mbtowc	マルチバイト文字をワイド文字に変換
qsort	配列をソート
realloc	確保済み領域の大きさを変更
strtod	文字列を double 型に変換
strtol	文字列を long 型に変換
strtoul	文字列を unsigned long 型に変換
wcstombs	ワイド文字列をマルチバイト文字列に変換
wctomb	ワイド文字をマルチバイト文字に変換

string.h

機能 文字列操作関数とメモリ操作関数のプロトタイプを宣言しています。プロトタイプを宣言している関数を以下に示します。

種別	関数名	機能
複写	strcpy	文字列の複写
	strncpy	文字列の複写(n 文字の複写)
連結	strcat	文字列の連結
	strncat	文字列の連結(n 文字の連結)
比較	strcmp	文字列の比較
	strcoll	文字列の比較(ロケール情報を使用)
	stricmp	文字列の比較(すべての英字は英大文字として扱う)
	strncmp	文字列の比較(n 文字の比較)
	strnicmp	文字列の比較(n 文字の比較、英字は英大文字として扱う)
検索	strchr	文字列の先頭より指定文字を検索
	strcspn	文字列より指定以外の文字列の長さを計算
	strpbrk	文字列より指定文字の検索
	strrchr	文字列の末尾より指定文字を検索
	strspn	文字列より指定文字列の長さを計算
	strstr	文字列より指定文字列の検索
	strtok	文字列より文字列を切り出す
長さ	strlen	文字列中の文字数を計算
変換	strerror	エラー番号を文字列に変換
	strxfrm	文字列を変換(ロケール情報を使用)
初期化	bzero	メモリ領域の初期化(ゼロクリア)
複写	bcopy	メモリ領域の複写
	memcpy	メモリ領域の複写(n 文字の複写)
	memset	メモリ領域の設定
比較	memcmp	メモリ量域の比較(n バイトの比較)
	memicmp	メモリ領域の比較(英文字は大文字として扱う)
検索	memchr	メモリ領域より文字を検索

time.h

機能 現在の暦時間を表現するための関数の宣言と型を定義しています。

E.2 標準関数リファレンス

本コンパイラの標準関数ライブラリの機能と詳細の仕様を説明します。

E.2.1 標準関数ライブラリの概要

本コンパイラは 119 個の標準関数ライブラリを用意しています。各関数は機能的に以下の 11 種類に分類されます。

(1) 文字列操作関数

文字列のコピー、比較等を行う関数です。

(2) 文字判定関数

アルファベット、10 進文字等の判定、及び大文字から小文字へ、小文字から大文字へ変換する関数です。

(3) 入出力関数

文字、文字列の入出力を行う関数です。中には、書式変換付きの入出力、及び文字列操作を行う関数も含まれています。

(4) メモリ管理関数

動的メモリ領域の確保、及び解放を行う関数です。

(5) メモリ操作関数

メモリ領域の複写、設定、及び比較を行う関数です。

(6) 実行制御関数

プログラムの実行を終了する関数と、現在実行中の関数から別の関数へジャンプするための関数です。

(7) 数学関数

sin、cos 等の演算を行う関数です。

- これらの関数は時間を要します。このため、監視タイマには十分注意してください。

(8) 整数算術関数

整数値に対しての演算を行う関数です。

(9) 文字列数値変換関数

文字列を数値に変換する関数です。

(10) 多バイト文字/多バイト文字列操作関数

多バイト文字/多バイト文字列を扱う関数です。

(11) 地域化関数

ロケールに関する関数です。

E.2.2 標準関数ライブラリ機能別一覧

(1) 文字列操作関数

文字列操作関数の一覧を【表E.2】に示します。

表E.2 文字列操作関数

種別	関数名	機能	リエントラント性
複写	strcpy	文字列の複写	○
	strncpy	文字列の複写(n 文字の複写)	○
連結	strcat	文字列の連結	○
	strncat	文字列の連結(n 文字の連結)	○
比較	strcmp	文字列の比較	○
	strcoll	文字列の比較(ロケール情報を使用)	○
	stricmp	文字列の比較(すべての英字は英大文字として扱う)	○
	strncmp	文字列の比較(n 文字の比較)	○
	strnicmp	文字列の比較(n 文字の比較、英字は英大文字として扱う)	○
検索	strchr	文字列の先頭より指定文字を検索	○
	strcspn	文字列より指定以外の文字列の長さを計算	○
	strpbrk	文字列より指定文字の検索	○
	strrchr	文字列の末尾より指定文字を検索	○
	strspn	文字列より指定文字列の長さを計算	○
	strstr	文字列より指定文字列の検索	○
	strtok	文字列より文字列を切り出す	X
長さ	strlen	文字列中の文字数を計算	○
変換	strerror	エラー番号を文字列に変換	X
	strxfrm	strxfrm	○

(2) 文字操作関数

文字操作関数の一覧を【表E.3】に示します。

表E.3 文字操作関数

関数名	機能	リエントラント性
isalnum	英数字の判定	○
isalpha	英字の判定	○
iscntrl	コントロール文字の判定	○
isdigit	数字の判定	○
isgraph	英数字、ブランク以外の文字判定	○
islower	英数字、ブランク以外の文字判定	○
isprint	ブランク文字を含む印字可能文字の判定	○
ispunct	区切り文字の判定	○
isspace	ブランク、タブ、改行の判定	○
isupper	英大文字の判定	○
isxdigit	16 進数字の判定	○
tolower	大文字から小文字への変換	○
toupper	小文字から大文字への変換	○

(3) 入出力関数

入出力関数の一覧を【表E.4】に示します。

表E.4 入出力関数

種別	関数名	機能	リentrant性
初期化	init	マイコンの入出力の初期化	X
	clearerr	エラー状態指示子を初期化(クリア)	X
入力	fgetc	一文字入力	X
	getc	一文字入力	X
	getchar	stdin からの一文字入力	X
	fgets	一行入力	X
	gets	stdin からの一文字入力	X
	fread	指定データ数入力	X
	scanf	stdin からの書式付き入力	X
	fscanf	書式付き入力	X
	sscanf	文字からの書式付きデータ入力	X
出力	fputc	一文字出力	X
	putc	一文字出力	X
	putchar	stdout への一文字出力	X
	fputs	一行出力	X
	puts	stdout への一文字出力	X
	fwrite	指定データ数出力	X
	perror	stdout へのエラーメッセージ出力	X
	printf	stdout への書式付き出力	X
	fflush	出力バッファのストリームをフラッシュ	X
	fprintf	書式付き出力	X
	sprintf	書式付き文字列設定	X
	vfprintf	ストリームへの書式付き出力	X
	vprintf	stdout への書式付き出力	X
	vsprintf	バッファへの書式付き出力	X
返還	ungetc	一文字入力の返還	X
判定	ferror	入出力エラーの判定	X
	feof	EOF(End Of File)の判定	X

(4) メモリ管理関数

メモリ管理関数の一覧を【表E.5】に示します。

表E.5 メモリ管理関数

関数名	機能	リentrant性
calloc	メモリの確保と 0(ゼロ)による初期化	X
free	メモリの解放	X
malloc	メモリの確保	X
realloc	確保済み領域の大きさを変更	X

(5) メモリ操作関数

メモリ操作関数の一覧を【表E.6】に示します。

表E.6 メモリ操作関数

種別	関数名	機能	リエントラント性
初期化	bzero	メモリ領域の初期化(ゼロクリア)	○
複写	bcopy	メモリ領域の複写	○
	memcpy	メモリ領域の複写(n文字の複写)	○
	memset	メモリ領域の設定	○
比較	memcmp	メモリ量域の比較(nバイトの比較)	○
	memicmp	メモリ領域の比較(英文字は大文字として扱う)	○
移動	memmove	文字列の領域を移動	○
検索	memchr	メモリ領域より文字を検索	○

(6) 実行制御関数

実行制御関数の一覧を【表E.7】に示します。

表E.7 実行制御関数

関数名	機能	リエントラント性
abort	プログラムの実行を終了	○
longjmp	大域ジャンプ	○
setjmp	大域ジャンプのためのスタック環境を設定	○

(7) 数学関数

数学関数の一覧表を【表E.8】に示します。

表E.8 数学関数

関数名	機能	リエントラント性
acos	逆コサインを計算	○
asin	逆サインを計算	○
atan	逆タンジェントを計算	○
atan2	逆タンジェントを計算	○
ceil	整数繰り上げ値を計算	○
cos	コサインを計算	○
cosh	双曲線コサインを計算	○
exp	指数関数を計算	○
fabs	倍精度浮動小数の絶対値を計算	○
floor	整数繰り下げ値を計算	○
fmod	剰余計算	○
frexp	浮動小数を仮数部と指数部に分割	○
labs	long 型整数の絶対値を計算	○
ldexp	浮動小数の中を計算	○
log	自然対数を計算	○
log10	常用対数を計算	○
modf	実数を仮数部と指数部に分割	○
pow	巾乗計算	○
sin	サインを計算	○
sinh	双曲線サインを計算	○
sqrt	数値の平方根を計算	○
tan	タンジェントを計算	○
tanh	双曲線タンジェントを計算	○

(8) 整数算術関数

整数算術関数の一覧表を【表E.9】に示します。

表E.9 整数算術関数

関数名	機能	リエントラント性
abs	整数の絶対値を計算	○
bsearch	配列内のバイナリサーチを行う	○
div	int 型整数の除算と剰余	○
labs	long 型整数の絶対値を計算	○
ldiv	long 型整数の除算と剰余	○
qsort	配列をソート	○
rand	疑似乱数を発生	○
srand	疑似乱数にシードを与える	○

(9) 文字列数値変換関数

文字列数値変換関数の一覧表を【表E.10】に示します。

表E.10 文字列数値変換関数

関数名	機能	リエントラント性
atof	文字列を double 型に変換	○
atoi	文字列を int 型に変換	○
atol	文字列を long 型に変換	○
strtod	文字列を double 型に変換	○
strtol	文字列を long 型に変換	○
strtou	文字列を unsigned long 型に変換	○

(10) 多バイト文字/多バイト文字列操作関数

多バイト文字/多バイト文字列操作関数の一覧表を【表E.11】に示します。

表E.11 多バイト文字/多バイト文字列操作関数

関数名	機能	リエントラント性
mblen	マルチバイト文字列の長さを計算	○
mbstowcs	マルチバイト文字列をワイド文字列に変換	○
mbtowc	マルチバイト文字をワイド文字に変換	○
wcstombs	ワイド文字列をマルチバイト文字列に変換	○
wctomb	ワイド文字をマルチバイト文字に変換	○

(11) 地域化関数

地域化関数の一覧表を【表E.12】に示します。

表E.12 地域化関数

関数名	機能	リエントラント性
localeconv	構造体 lconv を初期化	○
setlocale	プログラムのロケール情報の設定と検索	○

acos

数学関数

機能 逆コサインを計算します。

書式 `#include <math.h>`

```
double acos( x );
```

実現方法 関数

引数 `double x;` 実数

戻り値

- `x`の値が-1.0から1.0の範囲外の場合はエラーとして0を返します。
- それ以外の場合は、0から π ラジアン of 範囲の値を返します。

asin

数学関数

機能 逆サインを計算します。

書式 `#include <math.h>`

```
double asin( x );
```

実現方法 関数

引数 `double x;` 実数

戻り値

- `x`の値が-1.0から1.0の範囲外の場合はエラーとして0を返します。
- それ以外の場合は、 $-\pi/2$ から $\pi/2$ ラジアン of 範囲の値を返します。

atan

数学関数

- 機能** 逆タンジェントを計算します。
- 書式** `#include <math.h>`
- `double atan(x);`
- 実現方法** 関数
- 引数** `double x;`
- 戻り値** $-\pi/2$ から $\pi/2$ ラジアン¹の範囲の値を返します。

atan2

数学関数

- 機能** "x"と"y"の商の逆タンジェントを計算します。
- 書式** `#include <math.h>`
- `double atan2(x , y);`
- 実現方法** 関数
- 引数** `double x;` 実数
`double y;` 実数
- 戻り値** $-\pi$ から π ラジアン¹の範囲の値を返します。

atof

文字列数値変換関数

機能 文字列を浮動小数に変換します。

書式 #include <stdlib.h>

```
double atof( s );
```

実現方法 関数

引数 const char _far *s; 変換文字列へのポインタ

戻り値 文字列を倍精度浮動小数に返還した値を返します。

atoi

文字列数値変換関数

機能 文字列をint型整数に変換します。

書式 #include <stdlib.h>

```
int atoi( s );
```

実現方法 関数

引数 const char _far *s; 変換文字列へのポインタ

戻り値 文字列をint型整数に変換した値を返します。

atol**文字列数値変換関数**

- 機 能:** 文字列をlong型整数に変換します。
- 書 式:** `#include <stdlib.h>`
- `long atol(s);`
- 実現方法:** 関数
- 引 数:** `const char _far *s;` 変換文字列へのポインタ
- 戻り値:** 文字列をlong型整数に変換した値を返します。

B

bcopy

メモリ操作関数

機能 メモリ領域の複写を行います。

書式 `#include <string.h>`

```
void bcopy( src, dtop, size );
```

実現方法 関数

引数

<code>char _far *src;</code>	複写元のメモリ領域の先頭アドレス
<code>char _far *dtop;</code>	複写先のメモリ領域の先頭アドレス
<code>unsigned long size;</code>	複写するバイト数

戻り値 `dtop`で示される領域に、`src`で示される領域の先頭から`size`で指定されたバイト数分の内容を複写します。

bsearch

整数算術変換関数

機能 文字列を浮動小数に変換します

書式 `#include <stdlib.h>`
`void _far *bsearch(key, base, nelem, size, cmp);`

実現方法 関数

引数

<code>const void _far *key;</code>	検索キー
<code>const void _far *base;</code>	配列開始アドレス
<code>size_t nelem;</code>	要素数
<code>size_t size;</code>	各要素の大きさ
<code>int cmp();</code>	比較関数

戻り値

- 検索キーに等しい配列要素へのポインタを返します。
- 一致する要素がない場合はNULLポインタを返します。

解説 昇順にソート済みの配列内から指定された項目を検索します。

bzero

メモリ操作関数

機 能 メモリ領域の初期化(ゼロクリア)を行います。

書 式 #include <string.h>

```
void bzero( top, size );
```

実現方法 関数

引 数 char _far *top; ゼロクリアするメモリ領域の先頭アドレス
 unsigned long size; ゼロクリアするバイト数

戻り値 戻り値はありません。

解 説 topで示される領域の先頭アドレスからsizeで示されるバイト数分の内容を0で初期化します。

C

calloc

メモリ管理関数

- 機能** メモリの割り当てとゼロクリアを行います。
- 書式** `#include <stdlib.h>`
- `void *_far * calloc(n, size);`
- 実現方法** 関数
- 引数** `size_t n;` 要素の数
 `size_t size;` 要素の大きさをバイト数で表した値
- 戻り値** 指定した大きさの領域が確保できなかった場合、戻り値としてNULLを返します。
- 解説** ● 指定されたメモリを割り当てた後、ゼロクリアを行います。
 ● メモリ領域の大きさは2つの引数の積になります。
- 規則** メモリの確保規則については、`malloc`と同様です。

ceil

数学関数

- 機能** 整数繰り上げ値を返します。
- 書式** `#include <math.h>`
- `double ceil(x);`
- 実現方法** 関数
- 引数** `double x;` 実数
- 戻り値** xより大きい整数の中から最小の整数値をdouble型で返します。

clearerr

入出力関数

- 機能** ストリームのエラー状態指示子をクリアします。
- 書式** `#include <stdio.h>`
- `void clearerr(stream);`
- 実現方法** 関数
- 引数** `FILE _far *stream;` ストリームへのポインタ
- 戻り値** 戻り値はありません。
- 解説** エラー状態指示子とファイル終端状態指示子を正常値にリセットします。

COS

数学関数

- 機能** コサインを計算します。
- 書式** `#include <math.h>`
- `double cos(x);`
- 実現方法** 関数
- 引数** `double x;` 実数
- 戻り値** ラジアンを単位とする引数"x"のコサインを計算します。

cosh

数学関数

機 能: 双曲線コサインを計算します。

書 式: `#include <math.h>`

`double cosh(x);`

実現方法: 関数

引 数: `double x;` 実数

戻り値: "x"の双曲線コサインを計算します。

D

div

剰余算関数

機能 int型整数の除算を行います。

書式 #include <stdlib.h>

```
div_t div( number, denom );
```

実現方法 関数

引数 int number; 被除数
int denom; 除数

戻り値 "number"を"denom"で割った商と剰余を返します。

解説

- "number"を"denom"で割った商と剰余をdiv_tの構造体で返します。
- div_tはstdlib.h内で定義しています。この構造体は、int quot、int remというメンバーから構成されます。

E

exp

数学関数

機能	指数関数を計算します。
書式	<pre>#include <math.h> double exp(x);</pre>
実現方法	関数
引数	double x; 実数
戻り値	"x"の指数関数の計算結果を返します。

F

fabs

数学関数

機能 倍精度浮動小数の絶対値を計算します。

書式 `#include <math.h>`

```
double fabs( x );
```

実現方法 関数

引数 `double x;` 実数

戻り値 倍精度浮動小数の絶対値を返します。

feof

入出力関数

機能 ストリームのファイル状態指示子(EOF)を調べます。

書式 `#include <stdio.h>`

```
int feof( stream );
```

実現方法 マクロ

引数 `FILE _far *stream;` ストリームへのポインタ

戻り値

- ストリームがEOF場合、真(0以外)を返します。
- それ以外の場合、NULL(0)を返します。

解説

- ストリームがEOFまで読み込んだかどうか判定します。
- 0x1Aコードを終了コードとみなし、以降のデータを受け付けません。

ferror

入出力関数

- 機能** ストリームのエラー状態を調べます。
- 書式** `#include <stdio.h>`
- `int ferror(stream);`
- 実現方法** マクロ
- 引数** `FILE _far *stream;` ストリームへのポインタ
- 戻り値** ● ストリームがエラーの場合、真(0以外)を返します。
 ● それ以外の場合、NULL(0)を返します。
- 解説** ● ストリームがエラーかどうか判定します。
 ● 0x1Aコードを終了コードとみなし、以降のデータを受け付けません。

fflush

入出力関数

- 機能** 出力バッファをフラッシュします。
- 書式** `#include <stdio.h>`
- `int fflush(stream);`
- 実現方法** 関数
- 引数** `FILE _far *stream;` ストリームのポインタ
- 戻り値** 常に0を返します。

fgetc

入出力関数

- 機 能** ストリームから1文字を入力します。
- 書 式** `#include <stdio.h>`
- `int fgetc(stream);`
- 実現方法** 関数
- 引 数** `FILE _far *stream;` ストリームのポインタ
- 戻 り 値** ● 入力した1文字を返します。
 ● エラー又はストリームの終りの場合、EOFを返します。
- 解 説** ● ストリームから1文字を入力します。
 ● 0x1Aコードを終了コードとみなし、以降のデータを受け付けません。

fgets

入出力関数

機 能 ストリームから文字列を入力します。

書 式 `#include <stdio.h>`

```
char _far * fgets( buffer, n, stream );
```

実現方法 関数

引 数

<code>char _far *buffer;</code>	格納先のポインタ
<code>int n;</code>	最大文字数
<code>FILE _far *stream;</code>	ストリームのポインタ

戻り値

- 正常に入力できた場合、格納先のポインタ(引数で与えたポインタと同じ)を返します。
- エラー又はストリームの終わりの場合、NULLポインタを返します。

解 説

- 指定したストリームから文字列を入力し、バッファ(buffer)に格納します。入力を終了するのは、次の3つの場合です。
 - (1) 改行文字(`¥n`)を入力した場合
 - (2) `n-1`個の文字を入力した場合
 - (3) ストリームの終わりまで入力した場合
- 入力した文字列の最後には、ヌル文字(`¥0`)を付加します。
- 改行文字(`¥n`)は、そのまま格納します。
- `0x1A`コードを終了コードとみなし、以降のデータを受け付けません。

floor

数学関数

- 機 能:** 整数の繰り下げ値を計算します。
- 書 式:** `#include <math.h>`
- `double floor(x);`
- 実現方法:** 関数
- 引 数:** `double x;` 実数
- 戻り値:** 整数の繰り下げ値を**double**型で返します

fmod

数学関数

- 機 能:** 剰余計算を行います。
- 書 式:** `#include <math.h>`
- `double fmod(x ,y);`
- 実現方法:** 関数
- 引 数:** `double x;` 被除数
`double y;` 除数
- 戻り値:** "x"を"y"で割ったときの剰余を返します。

fprintf**入出力関数**

機 能 ストリームへの書式付き出力を行います。

書 式 `#include <stdio.h>`

```
int fprintf( stream, format, argument... );
```

実現方法 関数

引 数 `FILE _far *stream;` ストリームのポインタ
`const char _far *format;` 書式指定文字列のポインタ

戻り値

- 出力した文字数を返します。
- ハードウェアに起因するエラーの場合、EOFを返します。

解 説

- `format`の指定に従って`argument`を文字列に変換し、ストリームへ出力します。
- `format`の指定方法は`printf`と同様です。

fputc**入出力関数**

機 能 ストリームに1文字を出力します。

書 式 `#include <stdio.h>`

```
int fputc( c, stream );
```

実現方法 関数

引 数 `int c;` 出力する文字
`FILE _far *stream;` ストリームのポインタ

戻り値

- 正常に出力できた場合、出力した文字を返します。
- エラーの場合、EOFを返します。

解 説 ストリームに1文字を出力します。

fputs

入出力関数

機 能	ストリームへ文字列を出力します。	
書 式	<pre>#include <stdio.h> int fputs (str, stream);</pre>	
実現方法	関数	
引 数	<pre>const char _far *str;</pre>	出力する文字列のポインタ
	<pre>FILE _far *stream;</pre>	ストリームのポインタ
戻り値	<ul style="list-style-type: none">● 正常に出力できた場合、0を返します。● エラーの場合、0以外(EOF)を返します。	
解 説	ストリームへ文字列を出力します。	

fread

入出力関数

機 能 ストリームから固定長データを入力します。

書 式 `#include <stdio.h>`

```
size_t fread( buffer, size, count, stream );
```

実現方法: 関数

引 数:	<code>void _far *buffer;</code>	格納先のポインタ
	<code>size_t size;</code>	データ1項目のバイト数
	<code>size_t count;</code>	最大データ項目数
	<code>FILE _far *stream;</code>	ストリームのポインタ

戻り値 入力したデータ項目数を返します。

解 説

- ストリームからsizeのデータ長を持つデータをcountの項目数だけ入力し、バッファ(buffer)に格納します。
- count分のデータを入力する前にストリームの終わりになった場合、それまでに入力したデータ項目数を返します。
- 0x1Aコードを終了コードとみなし、以降のデータを受け付けません。

free

メモリ管理関数

機能 メモリの解放を行います。

書式 `#include <stdlib.h>`

```
void free( cp );
```

実現方法 関数

引数 `void _far *cp;` 解放するメモリ領域へのポインタ

戻り値 戻り値はありません。

解説 ● 以前に関数**malloc**、**calloc**によって割り当てられたメモリ領域の解放を行います。
● **NULL**を引数にした場合は処理を行いません。

frexp

数学関数

機能 浮動小数を仮数部と指数部に分割します。

書式 `#include <math.h>`

```
double frexp( x,prexp );
```

実現方法 関数

引数 `double x;` 浮動小数
`int _far *prexp;` 2を底とする指数を格納する領域へのポインタ

戻り値 "x"の仮数部を返します。

fscanf

入出力関数

機 能 ストリームからの書式付き入力を行います。

書 式 `#include <stdio.h>`

```
int fscanf( stream, format, argument... );
```

実現方法 関数

引 数 `FILE _far *stream;` ストリームのポインタ
`const char _far *format;` 書式指定文字列のポインタ

戻り値

- 各引数`argument`に格納したデータ数を返します。
- ストリームからデータとしてEOFを入力した場合、EOFを返します。

解 説

- `format`の指定に従ってストリームからの入力文字を変換し、各引数`argument`が示す引数に格納します。
- `argument`は各引数のポインタでなければいけません。
- 0x1Aコードを終了コードとみなし、以降のデータを受け付けません。
- `format`の指定方法は、`scanf`と同様です。

fwrite

入出力関数

機 能 ストリームへ固定長データを出力します。

書 式 `#include <stdio.h>`

```
size_t fwrite( buffer, size, count, stream );
```

実現方法 関数

引 数 `const void _far *buffer;` 出力データのポインタ
 `size_t size;` データ1項目のバイト数
 `size_t count;` 最大データ項目数
 `FILE _far *stream;` ストリームのポインタ

戻 り 値 出力したデータ項目数を返します。

解 説 ● ストリームへsizeのデータ長を持つデータをcountの項目数だけ出力します。
 ● count分のデータを出力する前にエラーになった場合は、それまでに出力したデータ項目数を返します。

G

getc

入出力関数

機能 ストリームから1文字を入力します。

書式 `#include <stdio.h>`

```
int getc( stream );
```

実現方法 マクロ

引数 `FILE _far *stream;` ストリームへのポインタ

戻り値

- 入力した1文字を返します。
- エラー又はストリームの終わりの場合、EOFを返します。

解説

- ストリームから1文字を入力します。
- 0x1Aコードを終了コードとみなし、以降のデータを受け付けません。

getchar

入出力関数

機能 `stdin`から1文字を入力します。

書式 `#include <stdio.h>`

```
int getchar( void );
```

実現方法 マクロ

引数 引数はありません。

戻り値

- 入力した1文字を返します。
- エラー又はストリームの終わりの場合、EOFを返します。

解説

- ストリーム(`stdin`)から1文字を入力します。
- 0x1Aコードを終了コードとみなし、以降のデータを受け付けません。

gets

入出力関数

機能 `stdin`から文字列を入力します。

書式 `#include <stdio.h>`

```
char _far * gets( buffer );
```

実現方法 関数

引数 `char _far *buffer;` 格納先のポインタ

戻り値 ● 正常に入力できた場合、格納先のポインタ(引数で与えたポインタと同じ)を返します。
● エラー又はストリームの終わりの場合、`NULL`ポインタを返します。

解説 ● `stdin`から文字列を1行入力し、バッファ(`buffer`)に格納します。
● 行末の改行文字(`\n`)は、ヌル文字(`\0`)に置き換えます。
● `0x1A`コードを終了コードとみなし、以降のデータを受け付けません。

init

入出力関数

機能 ストリームの初期化を行います。

書式 `#include <stdio.h>`

```
void init( void );
```

実現方法 関数

引数 引数はありません。

戻り値 戻り値はありません。

解説

- ストリームの初期化を行います。また関数内で`speed`、`init_pm`を呼び出して、UART、及びセン
トロニクス出力装置の初期設定を行います。
- `init`は通常、スタートアッププログラムから呼び出して使用します。

isalnum

文字操作関数

機能 英字(A~Z、a~z)、数字(0~9)を判定します。

書式 `#include <ctype.h>`

```
int isalnum( c );
```

実現方法 マクロ

引数 `int c;` 判定する文

戻り値

- 英字、又は数字の場合、0以外を返します。
- 英字、又は数字でない場合、0を返します。

解説 引数の文字を判定します。

isalpha

文字操作関数

機 能	英字(A~Z、a~z)を判定します。	
書 式	#include <ctype.h> int isalpha(c);	
実現方法	マクロ	
引 数	int c;	判定する文字
戻り値	<ul style="list-style-type: none">● 英字の場合、0以外を返します。● 英字でない場合、0を返します。	
解 説	引数の文字を判定します。	

isctrl

文字操作関数

機 能	コントロール文字(0x00~0x1f、0x7f)を判定します。	
書 式	#include <ctype.h> int isctrl(c);	
実現方法	マクロ	
引 数	int c;	判定する文字
戻り値	<ul style="list-style-type: none">● コントロール文字の場合、0以外を返します。● コントロール文字でない場合、0を返します。	
解 説	引数の文字を判定します。	

isdigit

文字操作関数

機能	数字(0~9)を判定します。	
書式	#include <ctype.h> int isdigit(c);	
実現方法	マクロ	
引数	int c;	判定する文字
戻り値	<ul style="list-style-type: none">● 数字の場合、0以外を返します。● 数字でない場合、0を返します。	
解説	引数の文字を判定します。	

isgraph

文字操作関数

機能	ブランク以外の文字(0x21~0x7e)を印字文字判定します。	
書式	#include <ctype.h> int isgraph(c);	
実現方法	マクロ	
引数	int c;	判定する文字
戻り値	<ul style="list-style-type: none">● 印字可能な場合、0以外を返します。● 印字不可の場合、0を返します。	
解説	引数の文字を判定します。	

islower

文字操作関数

機能	英小文字(a~z)を判定します。	
書式	#include <ctype.h> int islower(c);	
実現方法	マクロ	
引数	int c;	判定する文字
戻り値	<ul style="list-style-type: none">● 英小文字の場合、0以外を返します。● 英小文字でない場合、0を返します。	
解説	引数の文字を判定します。	

isprint

文字操作関数

機能	ブランク文字を含む文字(0x20~0x7e)の印字文字を判定します。	
書式	#include <ctype.h> int isprint(c);	
実現方法	マクロ	
引数	int c;	判定する文字
戻り値	<ul style="list-style-type: none">● 印字可能な場合、0以外を返します。● 印字不可の場合、0を返します。	
解説	引数の文字を判定します	

isupper

文字操作関数

- 機能** 英大文字 (A~Z) を判定します。
- 書式** `#include <ctype.h>`
- `int isupper(c);`
- 実現方法** マクロ
- 引数** `int c;` 判定する文字
- 戻り値**
- 英大文字の場合、0以外を返します。
 - 英大文字でない場合、0を返します。
- 解説** 引数の文字を判定します。

isxdigit

文字操作関数

- 機能** 16進文字(0~9、A~F、a~f)を判定します。
- 書式** `#include <ctype.h>`
- `int isxdigit(c);`
- 実現方法** マクロ
- 引数** `int c;` 判定する文字
- 戻り値**
- 16進文字の場合、0以外を返します。
 - 16進文字でない場合、0を返します。
- 解説** 引数の文字を判定します。

L

labs

整数算術関数

機能	long型整数の絶対値を計算します。	
書式	<pre>#include <stdlib.h> long labs(n);</pre>	
実現方法	関数	
引数	long n;	long型整数
戻り値	long型整数の絶対値(0からの距離)を返します。	

ldexp

数学関数

機能	浮動小数の巾を計算します。	
書式	<pre>#include <math.h> double ldexp(x,exp);</pre>	
実現方法	関数	
引数	double x;	実数
	int exp;	巾乗数
戻り値	"x"*(2の"exp"乗)を返します。	

ldiv

整数算術関数

機能 long型整数の除算を行います。

書式 #include <stdlib.h>

```
ldiv_t ldiv( number, denom );
```

実現方法 関数

引数 long number; 被除数
long denom; 除数

戻り値 "number"を"denom"で割った商と剰余を返します。

解説

- "number"を"denom"で割った商と剰余をldiv_tの構造体で返します。
- ldiv_tはstdlib.h内で定義しています。この構造体は、long quot,long remというメンバーから構成されます。

localeconv

地域化関数

機能 構造体lconvを初期化します。

書式 #include <locale.h>

```
struct lconv _far *localeconv( void );
```

実現方法 関数

引数 引数はありません。

戻り値 初期化された構造体lconvへのポインタを返します。

log

数学関数

機能 自然対数を計算します。

書式 `#include <math.h>`

```
double log( x );
```

実現方法 関数

引数 `double x;` 実数

戻り値 "x"の自然対数を返します。

解説 関数expの逆関数です。

log10

数学関数

機能 常用対数を計算します。

書式 `#include <math.h>`

```
double log10( x );
```

実現方法 関数

引数 `double x;` 実数

戻り値 "x"の常用対数を返します。

longjmp

実行制御関数

機能 関数呼び出し時の環境の回復を行います。

書式 `#include <setjmp.h>`

```
void longjmp( env, val );
```

実現方法 関数

引数 `jmp_buf env;` 環境を回復する領域へのポインタ
`int val;` `setjmp`の結果として返す値

戻り値 戻り値はありません。

解説

- "env"で示される領域から環境を回復します。
- プログラムの制御は、以前に`setjmp`関数を呼んだ次の文に移ります。
- "val"で指定した値は、`setjmp`関数の結果として返します。ただし、"val"が"0"の場合"1"に変換されます。

M

malloc

メモリ管理関数

機能 mallocメモリの割り当てを行います。

書式 #include <stdlib.h>

```
void _far * malloc( nbytes );
```

実現方法 関数

引数 size_t nbytes; 割り当てるメモリの大きさバイト数

戻り値 指定した大きさの領域が確保できなかった場合は戻り値としてNULLを返します。

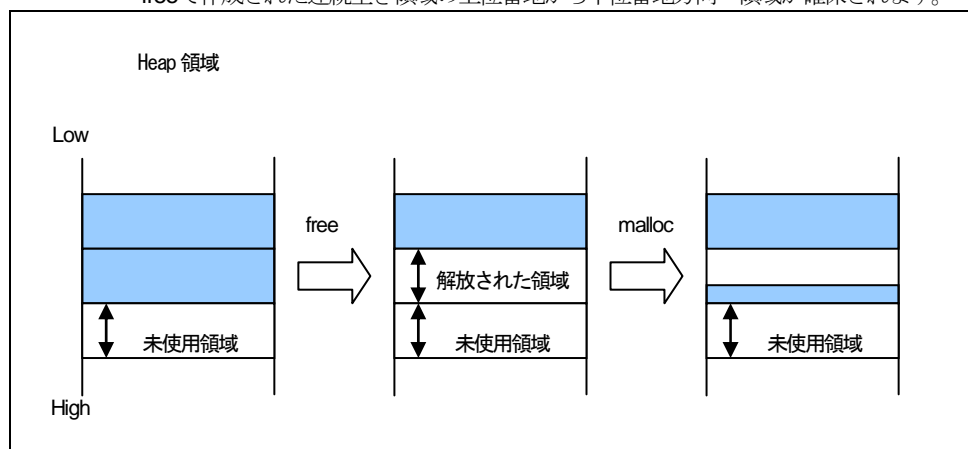
解説 動的なメモリ領域を割り当てます。

規則 mallocを行う場合、以下の(1)~(2)の順にチェックを行い、適合する箇所メモリを確保します。

(1) freeで解放された領域がある場合

- 確保するサイズがfreeで解放された領域より小さい場合

freeで作成された連続空き領域の上位番地から下位番地方向へ領域が確保されます。

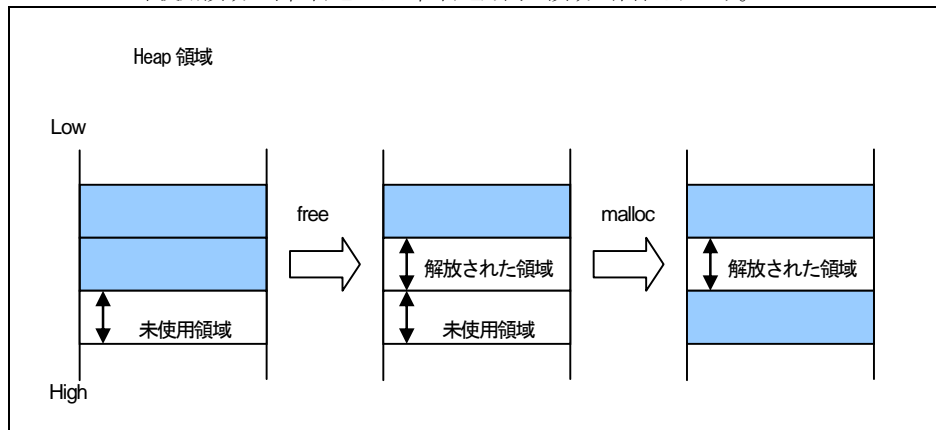


malloc

メモリ管理関数

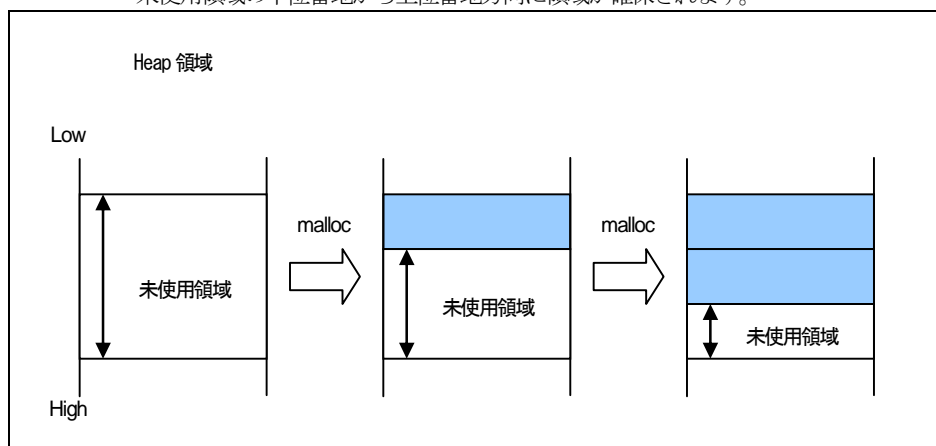
規 則:

- 確保するサイズがfreeで解放された領域よりも大きい場合
未使用領域の下位番地から上位番地方向に領域が確保されます。



(2) freeで解放された領域がない場合

- 確保可能な未使用領域が存在する場合
未使用領域の下位番地から上位番地方向に領域が確保されます。



- 確保可能な未使用領域が存在しない場合
メモリは確保せず、NULLを返します。

注意事項:

ガーベージコレクションは行っていません。したがって、小さな未使用領域が多く存在していても、指定した領域サイズ以上の未使用領域が無ければメモリを確保できず、NULLを返します。

mblen**多バイト文字/多バイト文字列操作関数**

- 機 能** マルチバイト文字列の長さを計算します。
- 書 式** `#include <stdlib.h>`
- ```
int mblen (s,n);
```
- 実現方法** 関数
- 引 数** `const char _far *s;` マルチバイト文字列へのポインタ  
`size_t n;` 検索バイト数
- 戻り値**
- `s`が正しいマルチバイト文字列をなしている場合はそのバイト数を返します。
  - `s`が正しいマルチバイト文字列をなしていない場合は-1を返します。
- 解 説** `s`がNULL文字を指している場合は0を返します。

---

**mbstowcs****多バイト文字/多バイト文字列操作関数**

- 機 能** マルチバイト文字列をワイド文字列に変換します。
- 書 式** `#include <stdlib.h>`
- ```
size_t mbstowcs( wcs,s,n );
```
- 実現方法** 関数
- 引 数** `wchar_t _far *wcs;` 変換ワイド文字列格納領域へのポインタ
`const char _far *s;` マルチバイト文字列へのポインタ
`size_t n;` 格納ワイド文字数
- 戻り値**
- 変換したマルチバイト文字列の文字数を返します。
 - 正しいマルチバイト文字列をなしていない場合は-1を返します。

mbtowc**多バイト文字/多バイト文字列操作関数**

機 能 マルチバイト文字をワイド文字に変換します。

書 式 `#include <stdlib.h>`

```
int mbtowc( wcs, s, n );
```

実現方法 関数

引 数

<code>wchar_t _far *wcs;</code>	変換ワイド文字列格納領域へのポインタ
<code>const char _far *s;</code>	マルチバイト文字列へのポインタ
<code>size_t n;</code>	検索バイト文字数

戻り値

- `s`が正しいマルチバイト文字をなしている場合は変換したワイド文字数を返します。
- `s`が正しいマルチバイト文字をなしていない場合は-1を返します。
- `s`がNULL文字の場合は0を返します。

memchr

メモリ操作関数

機能: メモリ領域より文字の検索を行います。

書式: `#include <string.h>`

```
void _far * memchr( s, c, n );
```

実現方法: 関数

引数: `const void _far *s;` 検索先のメモリ領域へのポインタ
`int c;` 検索する文字
`size_t n;` 検索するメモリ領域の大きさ

戻り値:

- 見つかった場合、指定された文字"**c**"の位置(ポインタ)を返します。
- メモリ領域中に文字"**c**"が見つからない場合は**NULL**を返します。

解説:

- "s"で示されるアドレスから始まる"n"で指定された大きさのメモリ領域から、"c"で示される文字を検索します。
- オプション-O[3~5]、-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

memcmp

メモリ操作関数

機 能 2つのメモリ領域の(nバイト)比較を行います。

書 式 #include <string.h>

```
int memcmp( s1, s2, n );
```

実現方法 関数

引 数

const void _far *s1;	比較する1番目のメモリ領域へのポインタ
const void _far *s2;	比較する2番目のメモリ領域へのポインタ
size_t n;	比較するバイト数

戻り値

- 戻り値==0 2番目のメモリ領域は等しい
- 戻り値>0 1番目のメモリ領域(s1)の方が大きい
- 戻り値<0 2番目のメモリ領域(s2)の方が大きい

解 説

- 2つのメモリ領域をnバイト分、バイト単位に比較します。
- オプション-O[3~5]、-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

memcpy

メモリ操作関数

機 能: メモリ領域の(nバイト)複写を行います。

書 式: `#include <string.h>`

```
void _far * memcpy( s1, s2, n );
```

実現方法: マクロ、関数

引 数: `void _far *s1;` 複写先のメモリ領域へのポインタ
`const void _far *s2;` 複写元のメモリ領域へのポインタ
`size_t n;` 複写するバイト数

戻り値: 複写先のメモリ領域へのポインタを返します。

解 説:

- 通常は、マクロが使用されます。ライブラリ関数を使用したい場合は、`#include <string.h>` の記述の後に、`#undef memcpy` と記述してください。
- "s1"で示される領域に、"n"で指定されたバイト数分"s2"で示されるメモリ領域より複写します。
- オプション-O[3~5]、-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

memicmp

メモリ操作関数

- 機 能** 2つのメモリ領域の(nバイト)比較を行います(英字はすべて大文字として扱います)。
- 書 式**

```
#include <string.h>

int memicmp( s1, s2, n);
```
- 実現方法** 関数
- 引 数**

```
char _far *s1;          比較する1番目のメモリ領域へのポインタ
char _far *s2;          比較する2番目のメモリ領域へのポインタ
size_t n;              比較するバイト数
```
- 戻り値**
 - 戻り値==0 2番目のメモリ領域は等しい
 - 戻り値>0 1番目のメモリ領域(s1)の方が大きい
 - 戻り値<0 2番目のメモリ領域(s2)の方が大きい
- 解 説**
 - 2つのメモリ領域を、nバイト分バイト単位に比較します。ただし、この時英字の大文字と小文字は区別せず小文字は大文字に変換して比較します。
 - オプション-O[3~5],-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

memmove**メモリ操作関数**

機能 メモリ領域を移動します。

書式 #include <string.h>

```
void _far * memmove( s1, s2, n );
```

実現方法 関数

引数 void _far *s1; 移動先へのポインタ
 const void _far *s2; 移動元へのポインタ
 size_t n; 移動バイト数

戻り値 移動先へのポインタを返します。

解説 オプション-O[3~5],-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

memset

メモリ操作関数

機能 メモリ領域を設定します。

書式 #include <string.h>

```
void _far * memset( s, c, n );
```

実現方法 マクロ、関数

引数 void _far *s; 設定先のメモリ領域への先頭ポインタ
 int c; 設定するデータ
 size_t n; 設定するバイト数

戻り値 設定先のメモリ領域への先頭ポインタを返します。

解説

- 通常は、マクロが使用されます。ライブラリ関数を使用したい場合は、#include <string.h> の記述の後に、#undef memcpy と記述してください。
- "s"で示される領域に、"n"で指定されたバイト数分"c"で指定されたデータを設定します。
- オプション-O[3~5]、-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

modf

数学関数

機能 実数を整数部と小数部に分割します。

書式 #include <math.h>

```
double modf ( val, pd );
```

実現方法 関数

引数 double val; 実数
 double _far *pd; 整数部格納領域へのポインタ

戻り値 実数の小数部を返します。

P

perror

入出力関数

機能 エラーメッセージをstderrに出力します。

書式 `#include <stdio.h>`

```
void perror( s );
```

実現方法 関数

引数 `const char _far *s;` メッセージの前につく文字列へのポインタ

戻り値 戻り値はありません。

pow

数学関数

機能 巾乗計算を行います。

書式 `#include <math.h>`

```
double pow( x,y );
```

実現方法 関数

引数 `double x;` 被乗数

`double y;` 巾乗数

戻り値 "x"の"y"乗を返します。

printf

入出力関数

機能 stdoutへの書式付き出力を行います。

書式 #include <stdio.h>

```
int printf( format, argument... );
```

実現方法 関数

引数 const char _far *format; 書式指定文字列のポインタ

formatで与えられる文字列の%以降の指定は、次の意味を持ちます。[]内は省略可能です。

書式: %[フラグ][最小フィールド幅][精度][修飾文字(l、L、又はh)]変換指定記号

書式例: %-05.8ld

戻り値

- 出力した文字数を返します。
- ハードウェアに起因するエラーの場合、EOFを返します。

解説

- formatの指定に従ってargumentを文字列に変換し、stdoutへ出力します。
- argumentにポインタを与える場合は、far型ポインタである必要があります。

(1) 変換指定記号

- d、i
引数の整数を符号付き10進数に変換します。
- u
引数の整数を符号なし10進数に変換します。
- o
引数の整数を符号なし8進数に変換します。
- x
引数の整数を符号なし16進数に変換します。0AH~0FHに小文字の"abcdef"を使用します。
- X
引数の整数を符号なし16進数に変換します。0AH~0FHに大文字の"ABCDEF"を使用します。
- c
引数の文字を1文字(ASCII文字)で出力します。
- s
引数の文字列far型ポインタ(char *)以降(ヌル文字'\0'又は精度数まで)を文字列に変換します。なお、wchar_t型の文字列は扱うことができません。

printf

入出力関数

解 説

- p
引数のポインタ(すべての型に対応)を24ビットアドレスで出力します。
 - n
n変換は引数の整数ポインタにそれまでに出力した文字数を格納します。引数は変換されません。
 - e
doubleの引数を指数形式に変換します。形式は[-]d.dddddde±ddです。
 - E
指数表示のeの代わりにEが使用される以外は、eと同じ書式です。
 - f
doubleの引数を[-]d.ddddd形式に変換します。
 - g
doubleの引数をe又はfにより指定される形式に変換します。通常はf型の変換を行います。指数部が-4以下、又は精度が指数値以下の場合、e型に変換されます。
 - G
指数表示のeの代わりにEが使用される以外は、gと同じ書式です。
 - -
変換の結果を最小フィールド幅内で左寄せします。デフォルトは右寄せです。
 - +
符号付きの変換結果に+又は-を付けます。デフォルトは、負の数のみ-を付けます。
 - ブランク"
デフォルトで符号付きの変換結果が符号なしになった場合、頭に空白"を付けます。
 - #
o変換では、頭に0を付けます。
x、X変換では0以外の時、頭に0x、0Xを付けます。
e、E、f変換では、常に小数点を付けます。
g、G変換では、常に小数点を付け、小数点以下の0も切り捨てずに出力します。
- (2) 最小フィールド幅
- 正の10進整数で最小のフィールド幅を指定します。
 - 変換結果の文字数が指定したフィールド数より少ない場合 左に埋め文字を挿入してフィールド幅を合わせます。
 - デフォルトの埋め文字は空白"です。頭に0を付けた整数でフィールド幅を指定した場合は、'0'を埋め文字とします。
 - フラグを指定した場合は、変換結果を左寄せし右に埋め文字を挿入します。この場合、埋め文字は常に空白"です。

printf

入出力関数

解 説

- 最小フィールド幅にアスタリスク(*)を指定した場合、引数の整数がフィールド幅を指定します。引数の値が負の場合、`-`フラグの後に正のフィールド幅を指定したことになります。
- (3) 精度
- !'の後に正の整数で指定し、!'のみの場合はゼロを指定したことになります。機能、及びデフォルト値は変換タイプにより異なります。
- 精度の指定がない浮動小数点型データの出力は、精度6で処理されます。ただし、精度が0のときは小数点は出力されません。
- d、i、o、u、x、X変換の場合
 - (1) 変換結果の桁数が指定した桁数より小さい場合、頭に'0'を挿入して桁数を合わせます。
 - (2) この指定が最小フィールド幅より大きい場合、こちらの指定が優先されます。
 - (3) この指定が最小フィールド幅より小さい場合、最小桁数の処理を行った後、フィールド幅の処理を行います。
 - (4) デフォルト値は1です。
 - (5) ゼロを最小桁数0で変換した場合、何も出力しません。
 - s変換の場合
 - (1) 最大文字数を表します。
 - (2) 変換結果が指定した文字数を越える場合 後が切り捨てられます。
 - (3) デフォルトには文字数制限がありません。
 - (4) 精度の指定にアスタリスク(*)を指定した場合
 - 引数の整数が精度を指定します。
 - (5) 引数の値が負の場合 精度の指定は無効になります。
 - e、E、f変換の場合

小数点の後にn個(nは精度)の数字を出力します。
 - g、G変換の場合

n個(nは精度)以上の有効数字を出力しません。
- (4) l、L¹、又はh
- lの場合、d、i、o、u、x、X、n変換をlong int又はunsigned long intの引数に対して行います。
 - hの場合、d、i、o、u、x、X変換をshort int又はunsigned short intの引数に対して行います。
 - l、hをd、i、o、u、x、X、n変換以外で指定した場合、指定が無視されます。
 - Lの場合、e、E、f、g、G変換をdoubleの引数に対して行います。

¹標準のC言語の仕様では、Lの場合、e、E、f、g変換をlong doubleの引数に対して行います。

本コンパイラでは、long doubleはdoubleとして扱いますので、Lを指定した場合doubleの引数として扱います。

putc

入出力関数

- 機 能** ストリームに1文字を出力します。
- 書 式** `#include <stdio.h>`
- `int putc(c, stream);`
- 実現方法** マクロ
- 引 数** `int c;` 出力する文字
 `FILE _far *stream;` ストリームのポインタ
- 戻り値** ● 正常に出力できた場合、出力した文字を返します。
 ● エラーの場合、EOFを返します。
- 解 説** ストリームに1文字を出力します。

putchar

入出力関数

- 機 能** `stdout`に1文字を出力します。
- 書 式** `#include <stdio.h>`
- `int putchar(c);`
- 実現方法** マクロ
- 引 数** `int c;` 出力する文字
- 戻り値** ● 正常に出力できた場合、出力した文字を返します。
 ● エラーの場合、EOFを返します。
- 解 説** `stdout`に1文字を出力します。

puts

入出力関数

機 能 `stdout`へ文字列を出力します。

書 式 `#include <stdio.h>`

```
int puts( str );
```

実現方法 マクロ

引 数 `char _far *str;` 出力する文字列のポインタ

戻 り 値 ● 正常に出力できた場合、0を返します。
● エラーの場合、-1(EOF)を返します。

解 説 ● `stdout`へ文字列を出力します。
● 文字列最後のヌル文字(`¥0`)は、改行文字(`¥n`)に置き換えて出力します。

Q

qsort

整数算術関数

機 能 配列をソートします。

書 式 `#include <stdlib.h>`

```
void qsort( base,nelen,size,cmp( e1,e2 ) );
```

実現方法: 関数

引 数:	<code>void _far *base;</code>	配列開始アドレス
	<code>size_t nelen;</code>	要素数
	<code>size_t size;</code>	各要素の大きさ
	<code>int cmp();</code>	要素を比較する関数

戻り値 戻り値はありません

解 説 配列をソートします。

R

rand

整数算術関数

機能 疑似乱数を発生します。

書式

```
#include <stdlib.h>

int rand( void );
```

実現方法 関数

引数 引数はありません。

戻り値

- srandで指定したseed乱数の系列を返します。
- 発生させる乱数は0~RAND_MAX間の値をとります。

realloc

メモリ管理関数

機能 確保済み領域の大きさを変更します。

書式

```
#include <stdlib.h>

void _far * realloc( cp, nbytes );
```

実現方法 関数

引数

void _far *cp;	変更前のメモリ領域へのポインタ
size_t nbytes;	変更するメモリ領域の大きさ(バイト数)

戻り値

- 大きさが変更された領域のポインタを返します。
- 指定した大きさの領域が確保できなかった場合はNULLを返します。

解説

- 関数mallocやcallocですでに確保済みの領域の大きさを変更します。
- 引数"cp"にすでに確保済みのポインタを指定し、引数"nbytes"で変更する大きさを指定します。

S

scanf

入出力関数

機能 `stdin`からの書式付き入力を行います。

書式

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

```
int scanf( format, argument... );
```

実現方法 関数

引数 `const char _far *format;` 書式指定文字列のポインタ

戻り値

`format`で与える文字列の%以降の指定は、次の意味を持ちます。[]内は省略可能です。

<p>書式: %[*][最大フィールド幅][修飾子(l、L、又はh)]変換指定記号 書式例: %*5ld</p>

戻り値

- 各引数`argument`に格納したデータ数を返します。
- `stdin`からデータとしてEOFを入力した場合、EOFを返します。

解説

- `format`の指定に従って`stdin`からの入力文字を変換し、各引数`argument`が示す変数に格納します。
- `argument`は各変数の`far`型ポインタでなければなりません。
- `c`、`[]`以外の変換において、先頭で入力したスペース文字は無視されます。
- `0x1A`コードを終了コードとみなし、以降のデータを受け付けません。

(1) 変換指定記号

- `d`
符号付きの10進数を変換します。対応する引数は整数へのポインタでなければいけません。
- `i`
符号付きの10進数、8進数、16進数の入力を変換します。8進数は先頭が0、16進数は先頭が0x、0Xで始まります。対応する引数は整数へのポインタでなければいけません。
- `u`
符号なし10進数を変換します。対応する引数は、符号なし整数へのポインタでなければいけません。
- `o`
符号付き8進数を変換します。対応する引数は整数へのポインタでなければいけません。

scanf

入出力関数

解 説

- x、X
符号付き16進数を変換します。0AH~0FHには大文字、小文字が使用できます。対応する引数は整数へのポインタでなければいけません。
 - s
ヌル文字'¥0'までの文字列を格納します。対応する引数はヌル文字'¥0'を含む文字列を格納するのに、十分な大きさを持つ文字配列を指すポインタでなければいけません。
最大フィールド幅に達して入力を中止した場合はそれまでの文字にヌル文字を付加した文字列を格納します。
 - c
文字を格納します。スペース文字は読み飛ばさずにそのまま格納します。
最大フィールド幅で2以上を指定した場合は、複数の文字を格納します。ただし、この場合はヌル文字'¥0'は付加しません。
対応する引数は文字列を格納するのに十分な大きさを持つ文字配列を指すポインタでなければいけません。
 - p
引数のポインタを出力します。
 - []
[]内の文字(複数指定可能)を入力している間、入力文字を格納します。[]内の文字以外を入力した場合、格納を中止します。
また[の次に^ (サーカムフレックス)を指定した場合は、^と]の間で指定した文字以外が入力許可文字となります。
指定した文字を入力した場合、格納を中止します。
対応する引数は自動的に付加するヌル文字'¥0'を含む文字列を格納するのに十分な大きさを持つ文字配列を指すポインタでなければいけません。
 - n
書式変換で既に読み込まれた文字数を格納します。対応する引数は整数へのポインタでなければいけません。
 - e、E、f、g、G
浮動小数点の形式に変換します。修飾子を指定したとき、対応する引数はdoubleへのポインタとなります。デフォルトはfloatへのポインタです。
- (2) *(データ格納の抑止指定)
- *が指定されている場合、変換したデータを引数に格納することを抑止します。
- (3) 最大フィールド幅
- 正の10進整数で最大入力文字数を指定します。1つの書式変換において、この文字数よりも多くの文字を読み込むことはありません。
 - 指定した文字数分の文字を読み込む以前に、スペース文字(関数isspace()で真となる文字)、又は要求する書式以外の文字を入力した場合は、その時点で読み込みを中止します。

scanf

入出力関数

- 解 説 (4) l、L、又はh
- l の場合、d、i、o、u、xの変換結果をlong int又はunsigned long intとして格納します。また、e、E、f、g、Gの変換結果をdoubleとして格納します。
 - h の場合、d、i、o、u、xの変換結果をshort int又はunsigned short intとして格納します。
 - l、h を d、i、o、u、x変換以外で指定した場合、指定が無視されます。
 - L の場合、e、E、f、g、Gの変換結果をfloatとして格納します。

setjmp

実行制御関数

- 機 能 関数呼び出し時の環境の退避を行います。
- 書 式 `#include <setjmp.h>`
- ```
int setjmp(env);
```
- 実現方法 関数
- 引 数 `jmp_buf env;` 環境を退避する領域へのポインタ
- 戻り値 `longjmp`の引数で与えられた数値を返します。
- 解 説 "env"で示される領域に環境の退避を行います。



---

**setlocale**

地域化関数

**機能** プログラムのロケール情報の設定と検索を行います。

**書式** `#include <locale.h>`

```
char _far *setlocale(category, locale);
```

**実現方法** 関数

**引数** `int category;` ロケール情報、検索部情報  
`const char _far *locale;` ロケール情報文字列へのポインタ

**戻り値**

- ロケール情報文字列へのポインタを返します。
- 設定、検索できない場合は**NULL**を返します。

---

**sin**

数学関数

**機能** サインを計算します。

**書式** `#include <math.h>`

```
double sin(x);
```

**実現方法** 関数

**引数** `double x;` 実数

**戻り値** ラジアンを単位とする"x"のサインを返します。

---

**sinh**

数学関数

機能 双曲線サインを計算します。

書式 `#include <math.h>`

```
double sinh(x);
```

実現方法 関数

引数 `double x;` 実数

戻り値 "x"の双曲線サインを返します。

---

**sprintf**

入出力関数

機能 文字列への書式付き出力を行います。

書式 `#include <stdio.h>`

```
int sprintf(pointer, format, argument...);
```

実現方法 関数

引数 `char _far *pointer;` 格納先のポインタ  
`const char _far *format;` 書式指定文字列のポインタ

戻り値 出力した文字数を返します。

解説

- `format`の指定に従って`argument`を文字列に変換し、`pointer`以降に格納します。
- `format`の指定方法は、`printf`と同様です。

---

**sqrt**

数学関数数

**機能** 数値の平方根を計算します。

**書式** `#include <math.h>`

```
double sqrt(x);
```

**実現方法** 関数

**引数** `double x;` 実数

**戻り値** 平方根値を返します。

---

**srand**

整数算術関数

**機能** 疑似乱数発生ルーチンにシードを与えます。

**書式** `#include <stdlib.h>`

```
void srand(seed);
```

**実現方法** 関数

**引数** `unsigned int seed;` 乱数の系列値

**戻り値** 戻り値はありません。

**解説** 引数"seed"を用いて、randによって与えられる疑似乱数系列を初期化します。

## sscanf

## 入出力関数

**機能** 文字列からの書式付き入力を行います。

**書式** #include <stdio.h>

```
int sscanf(string, format, argument...);
```

**実現方法** 関数

**引数** const char \_far \*string; 入力文字列のポインタ  
const char \_far \*format; 書式指定文字列のポインタ

**戻り値**

- 各引数argumentに格納したデータ数を返します。
- ヌル文字(¥0)をデータとして入力した場合、EOFを返します。

**解説**

- formatの指定に従って入力文字を変換し、各引数argumentが示す変数に格納します。
- argumentは各変数のfar型ポインタでなければいけません。
- formatの指定方法は、scanfと同様です。

## strcat

## 文字列操作関数

**機能** 文字列の連結を行います。

**書式** #include <string.h>

```
char _far * strcat(s1, s2);
```

**実現方法** 関数

**引数** char \_far \*s1; 連結先の文字列へのポインタ  
const char \_far \*s2; 連結する文字列へのポインタ

**戻り値** 連結された文字列領域(s1)へのポインタを返します。

**解説**

- "s1"で示される文字列の最後に、"s2"で示される文字列を連結します。
- 連結された文字列は、NULLで終了します。
- オプション-O[3~5]、-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strchr****文字列操作関数**

**機能** 文字列より文字の検索を行います。

**書式** `#include <string.h>`

`char _far * strchr( s, c );`

**実現方法** 関数

**引数** `const char _far *s;` 検索先の文字列へのポインタ  
`int c;` 検索する文字

**戻り値**

- 文字列"s"中で最初に現れた文字"c"の位置を返します。
- 文字列"s"が文字"c"を含まない場合はNULLを返します。

**解説**

- "s"で示される領域の先頭より、"c"で示される文字を検索します。
- `?%0`も検索対象とすることができます。
- オプション-O[3~5]、-OR及びOSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strcmp****文字列操作関数**

**機 能** 2つの文字列の比較を行います。

**書 式** `#include <string.h>`  
  
`int strcmp( s1, s2 );`

**実現方法** マクロ、関数

**引 数** `const char _far *s1;` 比較する1番目の文字列へのポインタ  
`const char _far *s2;` 比較する2番目の文字列へのポインタ

**戻り値**

- 戻り値==0 2つの文字列は等しい
- 戻り値>0 1番目の文字列(s1)の方が大きい
- 戻り値<0 2番目の文字列(s2)の方が大きい

**解 説**

- 通常は、マクロが使用されます。ライブラリ関数を使用したい場合は、`#include<string.h>` の記述の後に、`#undef strcmp` と記述してください。
- NULLで終了している2つの文字列をバイト単位に比較します。
- オプション-O[3~5]、-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strcoll****文字列操作関数**

**機 能:** ロケール情報を使用して2つの文字列を比較します。

**書 式:** `#include <string.h>`  
  
`int strcoll( s1, s2 );`

**実現方法:** 関数

**引 数:** `const char _far *s1;` 比較する1番目の文字列へのポインタ  
`const char _far *s2;` 比較する2番目の文字列へのポインタ

**戻り値:**

- 戻り値==0 2つの文字列は等しい
- 戻り値>0 1番目の文字列(s1)の方が大きい
- 戻り値<0 2番目の文字列(s2)の方が大きい

**解 説:** オプション-O[3~5],-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strcpy****文字列操作関数**

**機能** 文字列の複写を行います。

**書式** `#include <string.h>`

```
char _far * strcpy(s1, s2);
```

**実現方法** マクロ、関数

**引数** `char _far *s1;` 複写先の文字列へのポインタ  
`const char _far *s2;` 複写元の文字列へのポインタ

**戻り値** 複写先の文字列へのポインタを返します。

**解説**

- 通常は、マクロが使用されます。ライブラリ関数を使用したい場合は、`#include <string.h>` の記述の後に、`#undef strcpy` と記述してください。
- `s1` で示される領域に `s2` で示される (NULL で終了している) 文字列を複写します。
- 複写先の文字列は、NULL で終了します。
- オプション `-O[3~5]`、`-OR` 及び `-OS` を指定した場合は、最適化により関数のインライン展開を行う可能性があります。



## strcspn

## 文字列操作関数

|      |                                                                                                                                               |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| 機能   | 指定外文字列の長さを求めます。                                                                                                                               |
| 書式   | <pre>#include &lt;string.h&gt;  size_t strcspn( s1, s2 );</pre>                                                                               |
| 実現方法 | 関数                                                                                                                                            |
| 引数   | <pre>const char _far *s1;      検索先の文字列へのポインタ const char _far *s2;      検索する文字列へのポインタ</pre>                                                    |
| 戻り値  | 指定外文字列の長さを返します。                                                                                                                               |
| 解説   | <ul style="list-style-type: none"><li>● "s1"で示される領域の先頭より、"s2"で示される文字列に含まる文字以外で構成される最初の部分の文字列の長さを求めます。</li><li>● ¥0は検索対象とすることができません。</li></ul> |

## stricmp

## 文字列操作関数

|      |                                                                                                                                                        |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| 機能   | 2つの文字列の比較を行います(英字はすべて大文字として扱います)。                                                                                                                      |
| 書式   | <pre>#include &lt;string.h&gt;  int stricmp( s1, s2 );</pre>                                                                                           |
| 実現方法 | 関数                                                                                                                                                     |
| 引数   | <pre>char _far *s1;           比較する1番目の文字列へのポインタ char _far *s2;           比較する2番目の文字列へのポインタ</pre>                                                       |
| 戻り値  | <ul style="list-style-type: none"><li>● 戻り値==0    2つの文字列は等しい</li><li>● 戻り値&gt;0    1番目の文字列(s1)の方が大きい</li><li>● 戻り値&lt;0    2番目の文字列(s2)の方が大きい</li></ul> |
| 解説   | NULLで終了している2つの文字列をバイト単位に比較します。ただし、この時英字の大文字と小文字は区別せず小文字は大文字に変換して比較します。                                                                                 |

---

**strerror**

文字列操作関数

|      |                                                                        |
|------|------------------------------------------------------------------------|
| 機能   | エラー番号を文字列に変換します。                                                       |
| 書式   | <pre>#include &lt;string.h&gt;  char _far * strerror( errcode );</pre> |
| 実現方法 | 関数                                                                     |
| 引数   | int errcode;                      エラーコード                               |
| 戻り値  | エラーコードに対するメッセージ文字列へのポインタを返します。                                         |
| 解説   | stderrは静的な配列に対してポインタを返します。                                             |

---

**strlen****文字列操作関数**

|      |                                                           |
|------|-----------------------------------------------------------|
| 機能   | 文字列の長さを求めます。                                              |
| 書式   | <pre>#include &lt;string.h&gt;  size_t strlen( s );</pre> |
| 実現方法 | 関数                                                        |
| 引数   | <code>const char _far *s;</code> 長さを求める文字列へのポインタ          |
| 戻り値  | 文字列の長さを返します。                                              |
| 解説   | "s"で示される文字列(NULLまで)の長さを求めます。                              |

---

**strncat****文字列操作関数**

|      |                                                                                                                                                                                                                   |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 機能   | 文字列の(n文字)連結を行います。                                                                                                                                                                                                 |
| 書式   | <pre>#include &lt;string.h&gt;  char _far * strncat( s1, s2, n );</pre>                                                                                                                                           |
| 実現方法 | 関数                                                                                                                                                                                                                |
| 引数   | <code>char _far *s1;</code> 連結先の文字列へのポインタ<br><code>const char _far *s2;</code> 連結する文字列へのポインタ<br><code>size_t n;</code> 連結する文字数                                                                                    |
| 戻り値  | 連結された文字列領域へのポインタを返します。                                                                                                                                                                                            |
| 解説   | <ul style="list-style-type: none"><li>● "s1"で示される文字列の最後に、"n"で指定された文字数の文字を"s2"で示される文字列より連結します。</li><li>● 連結された文字列は、NULLで終了します。</li><li>● オプション-O[3~5]、-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。</li></ul> |

---

**strncmp**

文字列操作関数

**機 能** 2つの文字列の(n文字)比較を行います。

**書 式** #include <string.h>

```
int strncmp(s1, s2, n);
```

**実現方法** 関数

**引 数**

|                      |                   |
|----------------------|-------------------|
| const char _far *s1; | 比較する1番目の文字列へのポインタ |
| const char _far *s2; | 比較する2番目の文字列へのポインタ |
| size_t n;            | 比較する文字数           |

**戻り値**

- 戻り値==0 2つの文字列は等しい
- 戻り値>0 1番目の文字列(s1)の方が大きい
- 戻り値<0 2番目の文字列(s2)の方が大きい

**解 説**

- NULLで終了している2つの文字列を"n"文字分バイト単位に比較します。
- オプション-O[3~5]、-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

## strncpy

## 文字列操作関数

機能 文字列の複写を行います。

書式 `#include <string.h>`

```
char _far * strncpy(s1, s2, n);
```

実現方法 関数

引数

|                                   |               |
|-----------------------------------|---------------|
| <code>char _far *s1;</code>       | 複写先の文字列へのポインタ |
| <code>const char _far *s2;</code> | 複写元の文字列へのポインタ |
| <code>size_t n;</code>            | 複写する文字数       |

戻り値 複写先の文字列へのポインタを返します。

解説

- "s1"で示される領域に、"n"で指定された文字数の文字を、"s2"で示される文字列より複写します。ただし、この時"n"で指定された文字数より多い部分は複写されず、その後に`¥0`を付加することも行いません。逆に、"s2"の示す文字列の長さが"n"文字より短い場合は、複写された文字列の後に"n"文字になるまで`¥0`が付加されます。
- オプション`-O[3~5]`、`-OR`及び`-OS`を指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strnicmp****文字列操作関数**

**機 能:** 2つの文字列の(n文字)比較を行います(英字はすべて大文字として扱います)。

**書 式:** #include <string.h>

```
int strnicmp(s1, s2, n);
```

**実現方法:** 関数

**引 数:** char \_far \*s1;                    比較する1番目の文字列へのポインタ  
char \_far \*s2;                    比較する2番目の文字列へのポインタ  
size\_t n;                         比較する文字数

**戻り値:**

- 戻り値==0     2つの文字列は等しい
- 戻り値>0     2つの文字列は等しい
- 戻り値<0     2番目の文字列(s2)の方が大きい

**解 説:**

- NULLで終了している2つの文字列を"n"文字分バイト単位に比較します。ただし、この時英字の大文字と小文字は区別せず小文字は大文字に変換して比較します。
- オプション-O[3~5]、-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strpbrk****文字列操作関数**

**機能** 文字列より指定文字の検索を行います。

**書式** `#include <string.h>`

```
char _far * strpbrk(s1, s2);
```

**実現方法** 関数

**引数** `const char _far *s1;` 検索先の文字列へのポインタ  
`const char _far *s2;` 検索する文字の文字列へのポインタ

**戻り値**

- 見つかった場合、見つかった位置(ポインタ)を返します。
- 見つからない場合、**NULL**を返します。

**解説**

- "s1"で示される領域より、"s2"で示される文字列中の文字が含まれているか検索します。
- **¥0**は検索対象とすることができません。
- オプション**-O[3~5]**、**-OR**及び**OS**を指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strchr****文字列操作関数**

**機 能** 文字列より文字の検索を行います。

**書 式** `#include <string.h>`

`char _far * strchr( s, c );`

**実現方法** 関数

**引 数** `const char _far *s;` 検索先の文字列へのポインタ  
`int c;` 検索する文字

**戻り値**

- 文字列"s"中で最後に現れた文字"c"の位置を返します。
- 文字列"s"が文字"c"を含まない場合はNULLを返します。

**解 説**

- "s"で示される領域の末尾より、"c"で示される文字を検索します。
- ¥0も検索対象とすることができます。
- オプション-O[3~5]、-OR及びOSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。



## strspn

## 文字列操作関数

機能 指定文字列の長さを求めます。

書式 #include <string.h>

```
size_t strspn(s1, s2);
```

実現方法 関数

引数 const char \_far \*s1; 検索先の文字列へのポインタ  
const char \_far \*s2; 検索する文字列へのポインタ

戻り値 ● 指定文字列の長さを返します。

解説 ● "s1"で示される領域の先頭より、"s2"で示される文字列に含まれる文字のみで構成される最初の部分の長さを求めます。  
● %0は検索対象とすることができません。  
● オプション-O[3~5]、-OR及びOSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strstr****文字列操作関数**

**機能** 指定文字列の検索を行います。

**書式** `#include <string.h>`

```
char _far * strstr(s1, s2);
```

**実現方法** 関数

**引数** `const char _far *s1;` 検索先の文字列へのポインタ  
`const char _far *s2;` 検索する文字の文字列へのポインタ

**戻り値**

- 見つかった場合、見つかった位置(ポインタ)を返します。
- 見つからない場合、NULLを返します。

**解説**

- "s1"で示される領域の先頭より、"s2"で示される文字列が最初に現れた位置(ポインタ)を返します。
- オプション-O[3~5]、-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strtod****文字列操作関数**

**機能** 文字列を倍精度浮動小数に変換します。

**書式** `#include <string.h>`

```
double strtod(s, endptr);
```

**実現方法** 関数

**引数** `const char _far *s;` 変換文字列  
`char _far * _far *endptr;` 変換されなかった残りの文字列へのポインタ

**戻り値**

- 戻り値 == 0L 数を構成しません。
- 戻り値 != 0L 構成した数をdouble型で返します。

**解説** オプション-O[3~5]、-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strtok****文字列操作関数**

**機能** 文字列の切り出しを行います。

**書式** `#include <string.h>`

```
char _far * strtok(s1, s2);
```

**実現方法** 関数

**引数** `char _far *s1;` 切り出し先の文字列へのポインタ  
`const char _far *s2;` 区切り文字へのポインタ

**戻り値**

- 見つかった場合、切り出したトークンへのポインタを返します。
- 見つからない場合、**NULL**を返します。

**解説**

- 最初の呼び出しでは、最初の字句の先頭文字へのポインタを返します。このとき返される文字の最後には、**NULL**文字が書き込まれます。その後の呼び出し("s1"が**NULL**の場合)では、順次、次の字句が返されます。"s1"に字句がなくなると**NULL**を返します。
- オプション-O[3~5]、-OR及び-Osを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

---

**strtol****文字列数値変換関数**

**機能** 文字列をlong型数値に変換します。

**書式** #include <string.h>

```
long strtol(s, endptr, base);
```

**実現方法** 関数

**引数**

|                           |                                      |
|---------------------------|--------------------------------------|
| const char _far *s;       | 変換文字列                                |
| char _far * _far *endptr; | 変換されなかった残りの文字列へのポインタ                 |
| int base;                 | 読み込み数値の基数(0~36)。0の場合は整数定数の形式を読み込みます。 |

**戻り値**

- 戻り値 == 0L 数を構成しません。
- 戻り値 != 0L 構成した数をlong型で返します。

**解説** オプション-O[3~5]、-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

## strtol

## 文字列数値変換関数

- 機能** 文字列をunsigned long型数値に変換します。
- 書式**

```
#include <string.h>
```

```
unsigned long strtoul(s, endptr, base);
```
- 実現方法** 関数
- 引数**

```
const char _far *s;
```

 変換文字列  

```
char _far * _far *endptr;
```

 変換されなかった残りの文字列へのポインタ  

```
int base;
```

 読み込む数値の基数(0~36)。0の場合は整数定数の形式を読み込みます。
- 戻り値**
  - 戻り値 == 0L 数を構成しません。
  - 戻り値 != 0L 構成した数をunsigned long型で返します。
- 解説** オプション-O[3~5]、-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

## strxfrm

## 文字列操作関数

- 機能** ロケール情報を使用して文字列を変換します。
- 書式**

```
#include <string.h>
```

```
size_t strxfrm(s1, s2, n);
```
- 実現方法** 関数
- 引数**

```
char _far *s1;
```

 変換結果文字列格納領域へのポインタ  

```
const char _far *s2;
```

 変換元文字列へのポインタ  

```
size_t n;
```

 変換バイト数
- 戻り値** 変換されたバイト数を返します。
- 解説** オプション-O[3~5]、-OR及び-OSを指定した場合は、最適化によりコード効率の良い別の関数等を選択する可能性があります。

## T

## tan

数学関数

機能      タンジェントを計算します。

書 式:      `#include <math.h>`

```
double tan(x);
```

実現方法:      関数

引 数:      `double x;`                      実数

戻り値:      ラジアンを単位とする"x"のタンジェントを返します。

## tanh

数学関数

機能      双曲線タンジェントを計算します。

書 式:      `#include <math.h>`

```
double tanh(x);
```

実現方法:      関数

引 数:      `double x;`                      実数

戻り値:      ラジアンを単位とする"x"の双曲線タンジェントを返します。

---

**tolower**

文字操作関数

- 機能** 英大文字を英小文字に変換します。
- 書式** `#include <ctype.h>`
- ```
int tolower( c );
```
- 実現方法** マクロ
- 引数** `int c;` 変換する文字
- 戻り値**
- 引数が英大文字の場合、英小文字を返します。
 - それ以外は渡した引数を返します。
- 解説** 引数の英大文字を英小文字に変換します。

toupper

文字操作関数

- 機能** 英小文字を英大文字に変換します。
- 書式** `#include <ctype.h>`
- ```
int toupper(c);
```
- 実現方法** マクロ
- 引数** `int c;` 変換する文字
- 戻り値**
- 引数が英小文字の場合、大文字を返します。
  - それ以外は渡した引数を返します。
- 解説** 引数の英小文字を英大文字に変換します。

## U

## ungetc

入出力関数

機能 ストリームへ1文字戻します。

書式 `#include <stdio.h>`

```
int ungetc(c, stream);
```

実現方法 マクロ

引数 `int c;` 戻す文字  
`FILE _far *stream;` ストリームのポインタ

戻り値

- 正常な場合は、戻した1文字を返します。
- ストリームが書き込みモード、エラー、EOFの場合、又はEOFを戻そうとした場合、EOFを返します。

解説

- ストリームに1文字を戻します。
- 0x1Aコードを終了コードとみなし、以降のデータを受け付けません。



## V

## vfprintf

入出力関数

**機能** フォーマットを指定してテキストを指定ストリームに書き込みます。

**書式**

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

```
int vfprintf(stream, format, ap...);
```

**実現方法** 関数

**引数**

```
FILE _far *stream; ストリームのポインタ
const char _far *format; 書式指定文字列へのポインタ
va_list ap; 引数リストの先頭へのポインタ
```

**戻り値** 出力した文字数を返します。

**解説**

- フォーマットを指定してテキストを指定ストリームに書き込みます。
- 可変長引数にポインタを記述する場合は、**far**型ポインタでなければなりません。

---

**vprintf**

入出力関数

**機能** フォーマットを指定してテキストを**stdout**に書き込みます。

**書式** `#include <stdarg.h>`  
`#include <stdio.h>`

```
int vprintf(format, ap...);
```

**実現方法** 関数

**引数** `const char _far *format;` 書式指定文字列へのポインタ  
`va_list ap;` 引数リストへのポインタ

**戻り値** 出力した文字数を返します

**解説**

- フォーマットを指定してテキストを**stdout**に書き込みます。
- 可変長引数にポインタを記述する場合は、**far**型ポインタでなければなりません。

---

**vsprintf**

入出力関数

**機能** フォーマットを指定してテキストを指定バッファに書き込みます。

**書式** `#include <stdarg.h>`  
`#include <stdio.h>`

```
int vsprintf(s, format, ap...);
```

**実現方法** 関数

**引数** `char _far *s;` 格納先へのポインタ  
`const char _far *format;` 書式指定文字列へのポインタ  
`va_list ap;` 引数リストへの先頭へのポインタ

**戻り値** 出力した文字数を返します。

**解説** 可変長引数にポインタを記述する場合は、**far**型ポインタでなければなりません。

## W

## wcstombs

## 多バイト文字/多バイト文字列操作関数

**機能**           ワイド文字列をマルチバイト文字列に変換します。

**書式**           #include <stdlib.h>

```
size_t _far wcstombs(s, wcs, n);
```

**実現方法**       関数

**引数**           char \_far \*s;                   変換マルチバイト文字列格納領域へのポインタ  
 const wchar\_t \_far \*wcs;       ワイド文字列先頭へのポインタ  
 size\_t n;                       格納マルチバイト文字数

**戻り値**

- 正しく変換された場合は格納したマルチバイト文字数を返します。
- そうでない場合は-1を返します。

## wctomb

## 多バイト文字/多バイト文字列操作関数

**機能**           ワイド文字をマルチバイト文字に変換します。

**書式**           #include <stdlib.h>]

```
int wctomb(s, wchar);
```

**実現方法**       関数

**引数**           char \_far \*s;                   変換マルチバイト文字格納領域へのポインタ  
 wchar\_t wchar;                ワイド文字マルチバイト文字に含めたバイト数を返します。

**戻り値**

- マルチバイト文字に含めたバイト数を返します。
- 対応するマルチバイト文字が存在しない場合は-1を返します。
- ワイド文字が0の場合は0を返します。

## E.2.4 標準関数ライブラリの使用に関する注意事項

### (a) 標準ヘッダファイルに関する注意事項

標準関数ライブラリ中の関数を使用する時は、必ず指定された標準ヘッダファイルをインクルードしてください。インクルードしない場合、引数及び戻り値の整合がとれませんのでプログラムが正常に動作しないことがあります。

### (b) 標準ライブラリ関数の最適化に関する注意事項

最適化オプション-O[3~5]、-OS、-OR のいずれかを指定した場合、標準関数に関する最適化を行いません。この最適化は-Ono\_stdlib を指定することにより、抑止することができます。ユーザー関数として、標準ライブラリ関数と同名の関数を使用する場合は、この最適化を抑止してください。

#### (1) 関数のインライン埋め込み

関数strcpy及びmemcpyについて、【表E.13】の条件を満たす時、関数のインライン埋め込みを行いません。

表E.13 標準ライブラリ関数に対する最適化条件

| 関数名    | 最適化条件                                     | 記述例                                              |
|--------|-------------------------------------------|--------------------------------------------------|
| strcpy | 第1引数：far ポインタ<br>第2引数：文字列定数               | strcpy(str, "sample");                           |
| memcpy | 第1引数：far ポインタ<br>第2引数：far ポインタ<br>第3引数：定数 | memcpy(str, "sample", 6);<br>memcpy(str, fp, 6); |

### E.3 標準入出力関数ライブラリのカスタマイズ方法

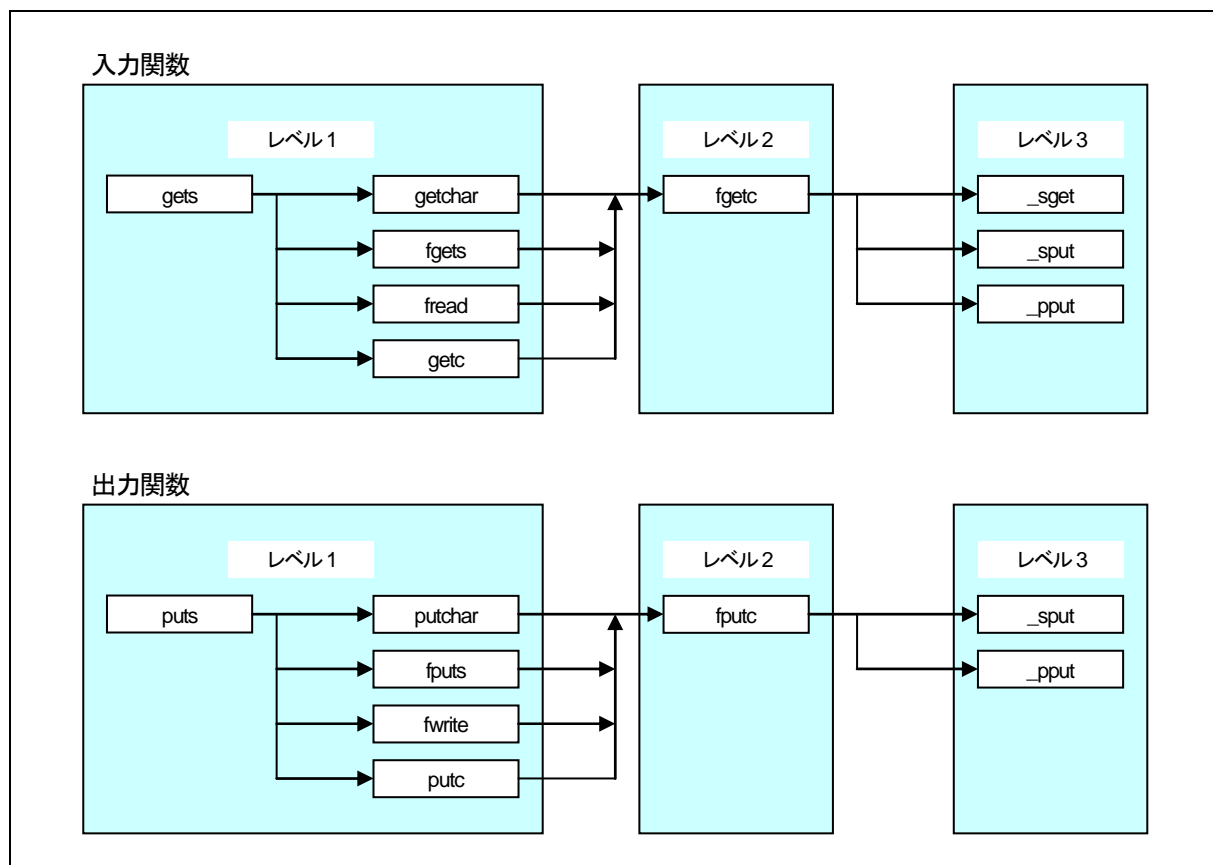
入出力関数として `scanf`、`printf` 等の高機能な関数ライブラリを用意しています。これらの関数は一般的に高水準入出力関数と呼ばれます。高水準入出力関数は、ハードウェア環境に依存した低水準入出力関数の組み合わせで実現しています。

アプリケーションプログラムにおいて、組み込むシステムのハードウェアによって入出力関数を変更する場合があります。このような場合、製品に添付しています標準関数ライブラリのソースファイルを変更することで、対処できます。

#### E.3.1 入出力関数の構成

入出力関数は、【図E.1】に示すようにレベル1の関数から下位の関数(レベル2→レベル3)を呼び出すことで機能を実現しています。例えば、`fgets`はレベル2の`fgetc`を呼び出し、更に`fgetc`はレベル3の関数を呼び出します。

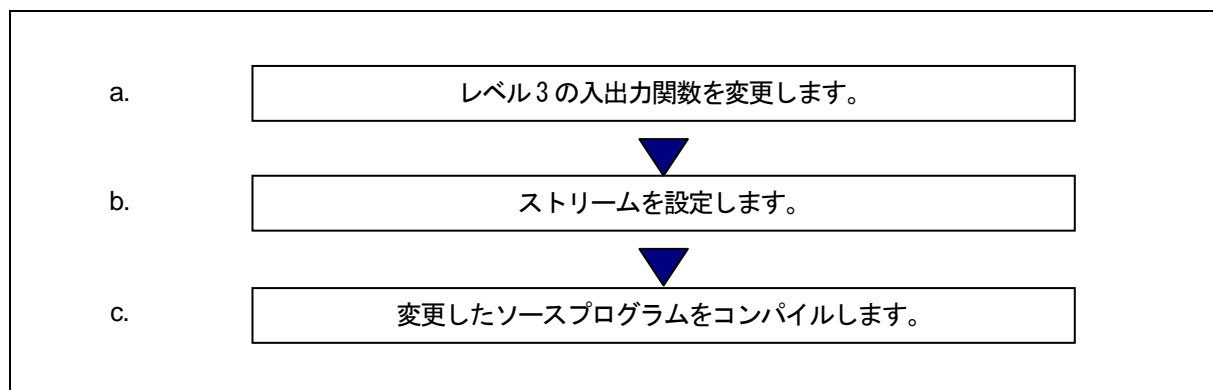
マイコンのハードウェア(入出力ポート)に依存しているのは最下位のレベル3の関数のみです。入出力関数をアプリケーションプログラムで使用する場合、必要に応じてレベル3の関数のソースファイルを書き換えることで、システムに順応した関数に変更できます。



図E.1 入出力関数の呼び出し関係図

### E.3.2 入出力関数の変更手順

入出力関数を組み込むシステムに合わせて変更する方法の概略を【図E.2】に示します。



図E.2 入出力関数の変更手順例

#### (a) レベル3の入出力関数の変更方法

レベル3の入出力関数は、R32C/100 シリーズの入出力ポートに対して1バイトの入出力を行う関数です。レベル3の入出力関数は、シリアル通信回路(UART)に対して入出力を行う `_sget`、`_sput` と、セントロニクス仕様の通信回路に対して入出力を行う `_pput` があります。

##### (1) 回路の諸設定

- クロック発信周波数：20MHz

##### (2) シリアル通信の初期設定

- UART1を使用
- ボーレート：9600bps
- データ長：8ビット
- パリティ：なし
- ストップビット：2ビット

\* これらのシリアル通信の初期設定は `_init` 関数(`init.c`)中で設定されています。

レベル3の入出力関数は、C言語で記述されたライブラリのソースファイル `device.c` に記述しています。レベル3の関数の仕様を【表E.14】に示します。

表E.14 レベル3の関数の仕様

| 入力関数                                                           | 引数        | 戻り値(int 型)                                    |
|----------------------------------------------------------------|-----------|-----------------------------------------------|
| <code>_sget</code><br><code>_sput</code><br><code>_pput</code> | なし        | 正常に入力できた場合は入力した文字を返します。<br>エラーの場合は EOF を返します。 |
| 出力関数                                                           | 引数(int 型) | 戻り値(int 型)                                    |
| <code>_sput</code><br><code>_pput</code>                       | 出力する文字    | 正常に出力できた場合は 1 を返します。<br>エラーの場合は EOF を返します。    |

シリアル通信は、R32C/100 シリーズが持つ2本のUARTの内UART1に設定しています。device.cでは、条件コンパイルコマンドでUART0を選択できるように記述しています。選択方法は、

- UART0を使用する場合..... #define UART0 1

をdevice.cファイルの先頭で記述するか、あるいは

- UART0を使用する場合..... -DUART0

をコンパイル時に指定します。

2本のUARTを使用する場合は、以下の手順で変更します。

- (1) device.cファイルの先頭に記述している条件コンパイルの記述を削除します。
- (2) #pragma EQUで定義されているUART0の特殊レジスタ名をUART1と異なった変数に書き換えます。
- (3) レベル3の関数\_sget、\_sputをUART0用にそれぞれ複製し、\_sget0、\_sput0等の異なった関数名に書き換えます。
- (4) speed関数もUART0用に複製し、speed0等の異なった関数名に書き換えます。

以上でdevice.cファイルの変更は終了します。

次に、入出力関数の初期設定を行っている\_init関数(init.c)を変更し、ストリームの設定を変更します。このストリームの設定方法は、次節で説明します。

## (b) ストリームの設定

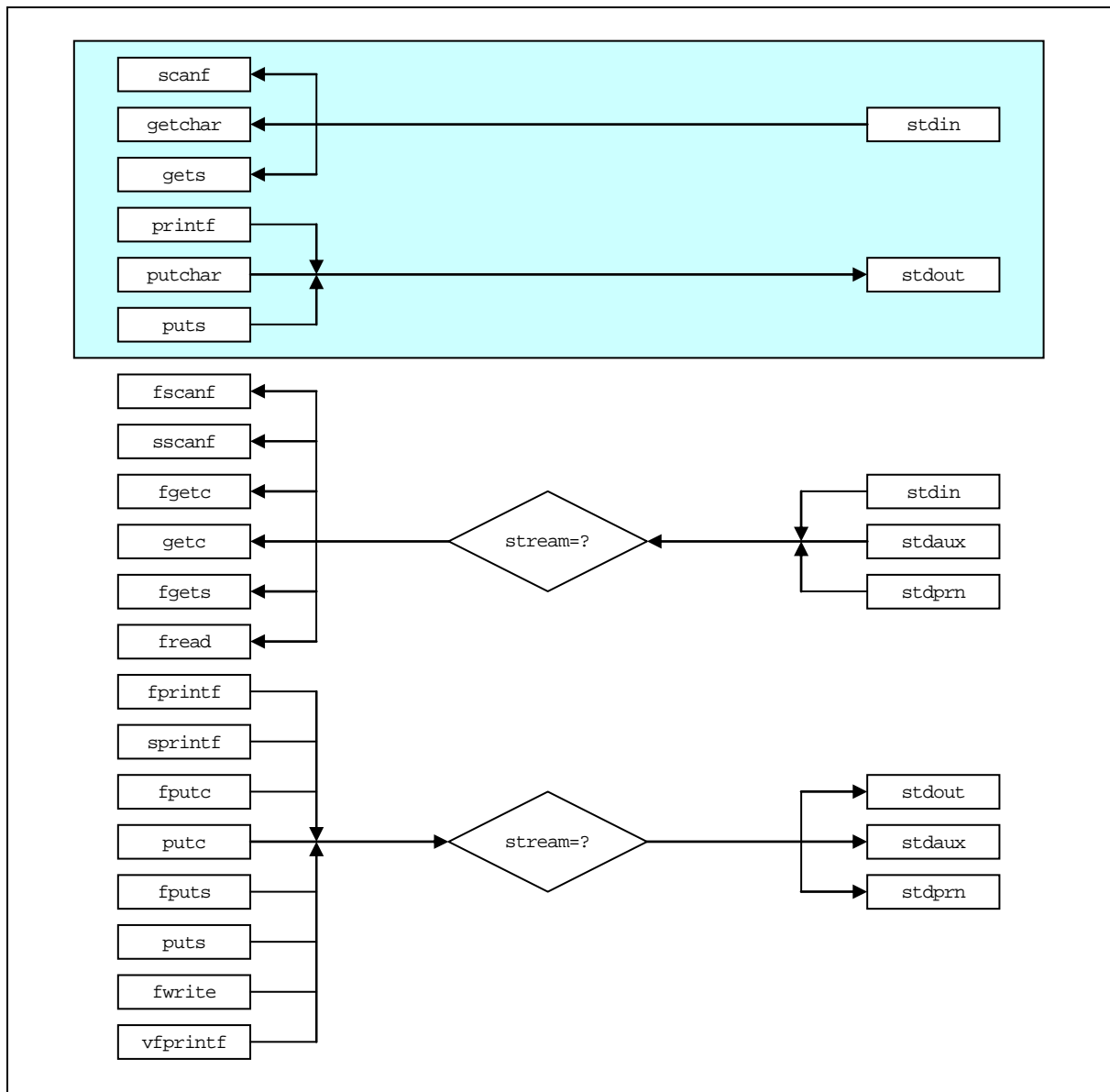
本コンパイラの標準ライブラリはstdin、stdout、stderr、stdaux、stdpmの5種類のストリーム情報を外部構造体として持っています。この外部構造体は標準ヘッダファイルstdio.h内で定義されており、各ストリームのモード情報(入力ストリームか出力ストリームを表すフラグ)やステータス情報(エラー又はEOFを表すフラグ)等を管理しています。

表E.15 ストリーム情報一覧表

| ストリーム情報 | 名称                                |
|---------|-----------------------------------|
| stdin   | 標準入力                              |
| stdout  | 標準出力                              |
| stderr  | 標準エラー出力 (stderrはstdoutに定義されています。) |
| stdaux  | 標準補助入出力                           |
| stdpm   | 標準プリンタ出力                          |

本コンパイラの標準ライブラリ関数中、【図E.3】の網掛部に示す関数の対応するストリームは標準入力(stdin)又は標準出力(stdout)に固定しています。これらの関数に関しては、ストリームを変更することはできません。なお、stderrはstdoutに#defineで定義されています。

ストリームは、fgetc、fputc等の引数としてストリームへのポインタを指定できる関数のみ変更することができます。



図E.3 関数とストリームの相互関係図

【図E.4】にstdio.h内のストリーム定義部を示します。



```

/*****
*
* standard I/O header file
*
* (省略)
*
typedef struct _iobuf {
 char _buff; /* Store buffer for ungetc */ ← (1)
 int _cnt; /* Strings number in _buff(1 or 0) */ ← (2)
 int _flag; /* Flag */ ← (3)
 int _mod; /* Mode */ ← (4)
 int (*_func_in)(void); /* Pointer to one byte input function */ ← (5)
 int (*_func_out)(int); /* Pointer to one byte output function */ ← (6)
} FILE;
#define _IOBUF_DEF
 :
 (省略)
 :
extern FILE _iob[];
#define stdin (&_iob[0]) /* Fundamental input */
#define stdout (&_iob[1]) /* Fundamental output */
#define stdaux (&_iob[2]) /* Fundamental auxiliary input output */
#define stdprn (&_iob[3]) /* Fundamental printer output */

#define stderr stdout /* NC no-support */

/*****
*
*****/
#define _IOREAD 1 /* Read only flag */
#define _IOWRT 2 /* Write only flag */
#define _IOEOF 4 /* End of file flag */
#define _IOERR 8 /* Error flag */
#define _IORW 16 /* Read and write flag */
#define _NFILE 4 /* Stream number */
#define _TEXT 1 /* Text mode flag */
#define _BIN 2 /* Binary mode flag */

 (以降省略)
 :

```

図E.4 stdio.h内のストリーム定義部 (stdio.h)

【図E.4】に示しましたファイル構造体の要素を以下に説明します。説中の (1) ~ (6) は、【図E.4】中の (1) ~ (6) に対応しています。

## (1) char \_buff

関数scanf、fscanfでは入力の際に1文字分の先読みを行っています。

先読みした文字が不要な場合は関数ungetcを呼び出して、先読みした文字をこの変数に格納します。

入力関数はこの変数にデータが存在する場合はこのデータを入力データとします。

## (2) int \_cnt

\_buffのデータ数を格納します(0もしくは1)。

- (3) `int _flag`  
読み込み専用フラグ(`_IOREAD`)、書き込み専用フラグ(`_IOWRT`)、読み書き両用フラグ(`_IORW`)、エンドオブファイルフラグ(`_IOEOF`)、エラーフラグ(`_IOERR`)のフラグを格納します。
- `_IOREAD`、`_IOWRT`、`_IORW`  
ストリームの動作モードを指定するフラグです。これらのフラグは、ストリームの初期化部分で設定します。
  - `_IOEOF`、`_IOERR`  
入出力関数内でEOF、エラーの発生に応じて設定されます。
- (4) `int _mod`  
テキストモード(`_TEXT`)、バイナリモード(`_BIN`)を表すフラグを格納します。
- テキストモード  
入出力データのエコーバック、文字変換を行います。エコーバック、文字変換の詳細については、関数`fgetc`、`fputc`のソースプログラム(`fgetc.c`、`fputc.c`)を参照ください。
  - バイナリモード  
入出力データを無変換で扱います。これらのフラグはストリームの初期化部分で設定します。
- (5) `int (*_func_in)()`  
ストリームが読み込みモード(`_IOREAD`)又は読み書き両用モード(`_IORW`)の場合、レベル3入力関数のポインタを格納します。それ以外の場合はNULLポインタを格納します。  
この情報をもとに、レベル2入力関数はレベル3入力関数を間接呼び出して呼び出します。
- (6) `int (*_func_out)()`  
ストリームが書き込みモード(`_IOWRT`)の場合、レベル3出力関数のポインタを格納します。また、ストリームが入力可能な場合(`_IOREAD`又は`_IORW`)で、かつテキストモードの場合、エコーバックするためのレベル3出力関数のポインタを格納します。それ以外の場合は、NULLポインタを格納します。  
この情報をもとに、レベル2入力関数はレベル3入力関数を間接呼び出して呼び出します。

ストリームの初期化では `char _buff` 以外のすべての要素に値を設定してください。

NC100 の製品に含まれる標準ライブラリファイルでは、関数 `_init` でストリームの初期化を行っています。関数 `_init` は、スタートアッププログラム `ncrt0.a30` から呼び出されています。【図E.5】に `_init` 関数のソースプログラムを示します。

```
#include <stdio.h>

FILE _iob[4];

void _init(void);

void _init(void)
{
 stdin->_cnt = stdout->_cnt = stdaux->_cnt = stdprn->_cnt = 0;
 stdin->_flag = _IOREAD;
 stdout->_flag = _IOWRT;
 stdaux->_flag = _IORW;
 stdprn->_flag = _IOWRT;

 stdin->_mod = _TEXT;
 stdout->_mod = _TEXT;
 stdaux->_mod = _BIN;
 stdprn->_mod = _TEXT;

 stdin->_func_in = _sget;
 stdout->_func_in = NULL;
 stdaux->_func_in = _sget;
 stdprn->_func_in = NULL;

 stdin->_func_out = _sput;
 stdout->_func_out = _sput;
 stdaux->_func_out = _sput;
 stdprn->_func_out = _pput;

#ifdef UART0
 speed(_96, _B8, _PN, _S2);
#else /* UART1 : default */
 speed(_96, _B8, _PN, _S2);
#endif
 init_prn();
}
```

図E.5 \_init関数のソースファイル (init.c)

R32C/100 シリーズの2本のUARTを使用するシステムでは、\_init関数を以下の手順で変更します。前節では、device.cソースファイル中でUART0用の関数を仮に\_sget0、\_sput0、speed0と設定しました。

- (1) UART0用のストリームには、標準補助入出力(stdaux)を使用します。
- (2) 標準補助入出力に対するフラグ(\_flag)、モード(\_mod)をシステムに合わせて設定します。
- (3) 標準補助入出力に対するレベル3関数のポインタを設定します。
- (4) speed関数に対する条件コンパイルコマンドを削除し、UART0用のspeed0関数に書き換えます。

以上の設定で、2本のUARTが使用できます。ただし、標準入出力のストリームを使用する関数はUART0が使用する標準補助入出力に対して使用することができません。したがって、関数の引数にストリームを記述できる関数で使用するください。【図E.6】に\_init関数の変更例を示します。

```

void _init(void)
{
 :
 (省略)
 :
 stdaux->_flag = _IORW; ← (2) 読み書き両用モードに設定
 :
 (省略)
 :
 stdaux->_mod = _TEXT; ← (2) テキストモードに設定
 :
 (省略)
 :
 stdaux->_func_in = _sget0; ← (3) UART0 用のレベル 3 の入力関数を指定
 :
 (省略)
 :
 stdaux->_func_out = _sput0; ← (3) UART0 用のレベル 3 の出力関数を指定
 :
 (省略)
 :
 speed(_96, _B8, _PN, _S2); ← (4) UART0 用の speed 関数を指定
 init_prn();
}

```

図E.6 \_init関数の変更例

### (c) 変更したソースプログラムの組み込み

変更したソースファイルをシステムに組み込む方法として、以下の2通りの方法があります。

- (1) 変更した関数のソースファイルのオブジェクトファイルをリンク時に指定します。
- (2) 製品に同封の実行手順ファイル(makefile.dos)を使用してライブラリファイルを更新します。

手順(1)の場合、リンク時に指定した関数が有効となり、ライブラリファイル内の同一名の関数は組み込まれません。

(1)の方法を【図E.7】に、(2)の方法を【図E.8】にそれぞれ示します。

```
% nc100 -c -g -osample ncrt0.a30 device.r30 init.r30 sample.c<RET>
```

※この例は、device.c と init.c を変更したときの記述方法です。

図E.7 変更したソースプログラムを直接リンクする方法

```
% make <RET>
```

図E.8 変更したソースプログラムを基にライブラリを更新する方法

## 付録F エラーメッセージ一覧表

NC100 が出力するすべてのエラーメッセージとワーニングメッセージ、そのメッセージに対する対処方法を説明します。

### F.1 メッセージの出力形式

NC100 は、処理中にエラーを検出した場合、エラーメッセージを画面に表示後、コンパイルを中止します。

【図F.1】～【図F.3】にエラーメッセージとワーニングメッセージの出力形式を示します。

```
nc100:[エラーメッセージ]
```

図F.1 コンパイルドライバnc100のエラー出力形式

```
[Error(cpp100.エラー番号):ファイル名,行番号]エラーメッセージ
[Error(ccom):ファイル名,行番号]エラーメッセージ
[Fatal(ccom):ファイル名,行番号]エラーメッセージ ← 致命的エラーの場合のメッセージです。
通常、このようなエラーメッセージは出力されませんが、
万一発生した場合は表示内容をご連絡ください。
```

図F.2 各コマンドのエラー出力形式

```
[Warning(cpp100.ワーニング番号):ファイル名,行番号]ワーニングメッセージ
[Warning(ccom):ファイル名,行番号]ワーニングメッセージ
```

図F.3 各コマンドのワーニング出力形式

## F.2 nc100 エラーメッセージ

【表F.1】と【表F.2】にコンパイルドライバnc100が出力するエラーメッセージとその内容及び対処方法を示します。

表F.1 nc100エラーメッセージ一覧表 (1/2)

| エラーメッセージ                                  | エラー内容と対策                                                                                                                                                                        |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Arg list too long                         | <ul style="list-style-type: none"> <li>各処理系を起動するときのコマンドラインがシステムで定義された文字数を超えています。</li> </ul> ⇒ システムで定義された文字数を超えないようにコンパイルオプションを指定してください。各処理系のコマンドラインは、コンパイルオプション-vで確認することができます。  |
| Cannot analyze error                      | <ul style="list-style-type: none"> <li>通常は発生しません(内部エラーです)。</li> </ul> ⇒ 弊社までご連絡ください。                                                                                            |
| command-file line characters exceed 2048. | <ul style="list-style-type: none"> <li>コマンドファイルの1行の文字数が2048文字を越えています。</li> </ul> ⇒ コマンドファイルの1行の文字数を2048文字以下にしてください。                                                             |
| Core dump(command_name)                   | <ul style="list-style-type: none"> <li>処理系がCore Dumpを起こしました。カッコ内はCore dumpを起こした処理系です。</li> </ul> ⇒ 各処理系が正しく実行されていません。環境変数又は各処理系が存在するディレクトリを確認してください。その上で正しく起動しない場合は、弊社にご連絡ください。 |
| Exec format error                         | <ul style="list-style-type: none"> <li>各処理系の実行ファイルが壊れています。</li> </ul> ⇒ 再度、インストールしなおしてください。                                                                                     |
| Ignore option '-?'                        | <ul style="list-style-type: none"> <li>NC100で使用できないコンパイルオプションオプション-?を使用しています。</li> </ul> ⇒ 正しいコンパイルオプション指定してください。                                                               |
| illegal option                            | <ul style="list-style-type: none"> <li>-as100 や-ln100 などの各処理系に指示するオプションが100文字を越えました。</li> </ul> ⇒ 各処理系に指示するオプションは99文字までにしてください。                                                 |
| Invalid argument                          | <ul style="list-style-type: none"> <li>通常は発生しません(内部エラーです)。</li> </ul> ⇒ 弊社までご連絡ください。                                                                                            |
| Invalid option '-?'                       | <ul style="list-style-type: none"> <li>コンパイルオプション-?に必要なパラメータがありません。</li> </ul> ⇒ コンパイルオプション-?の次に必要なパラメータを指定してください。                                                              |
| Invalid option '-o'                       | <ul style="list-style-type: none"> <li>コンパイルオプション-oの次に出力ファイル名がありません。</li> </ul> ⇒ 出力ファイル名を指定してください。ファイル名は拡張子を指定しないでください。                                                        |
| Invalid suffix '.xxx'                     | <ul style="list-style-type: none"> <li>NC100が認識できないファイル拡張子(c, .i, .a30, .r30, .x30 以外の拡張子)を使用しています。</li> </ul> ⇒ 正しい拡張子でファイルを指定してください。                                          |
| No such file or directory                 | <ul style="list-style-type: none"> <li>各処理系が実行できません。</li> </ul> ⇒ 各処理系が格納されているディレクトリを環境変数で正しく設定しているかを確認してください。                                                                  |
| Not enough core                           | <ul style="list-style-type: none"> <li>スワップ領域が不足しています。</li> </ul> ⇒ スワップ領域を増やしてください。                                                                                            |

表F.2 nc100エラーメッセージ一覧表 (2/2)

| エラーメッセージ                | エラー内容と対策                                                                                                                                              |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| Permission denied       | <ul style="list-style-type: none"> <li>各処理系が実行できません。</li> </ul> ⇒ 各処理系のパーミッションを確認してください。又、パーミッションが正しい場合は、各処理系が格納されているディレクトリを環境変数で正しく設定しているかを確認してください。 |
| can't open command file | <ul style="list-style-type: none"> <li>@で指定されたコマンドファイルがオープンできません。</li> </ul> ⇒ 正しいファイル名を指定してください。                                                     |
| too many options        | <ul style="list-style-type: none"> <li>指定されたコンパイルオプションの数が100を超えています。</li> </ul> ⇒ コンパイルオプションは99文字までにしてください。                                           |
| Result too large        | <ul style="list-style-type: none"> <li>通常は発生しません(内部エラーです)。</li> </ul> ⇒ 弊社までご連絡ください。                                                                  |
| Too many open files     | <ul style="list-style-type: none"> <li>通常は発生しません(内部エラーです)。</li> </ul> ⇒ 弊社までご連絡ください。                                                                  |

## F.3 cpp100 エラーメッセージ

【表F.3】～【表F.5】にプリプロセッサcpp100が出力するエラーメッセージとその内容及び対処方法を示します。

表F.3 cpp100エラーメッセージ一覧表 (1/3)

| 番号 | エラーメッセージ                        | エラー内容と対策                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | illegal command option          | <ul style="list-style-type: none"> <li>• 入力ファイル名を2回指定しています。</li> <li>⇒ 入力ファイル名の指定を1つにしてください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• 入力ファイル名と出力ファイル名が同じ名前です。</li> <li>⇒ 出力ファイル名は入力ファイル名と違う名前を指定してください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• 出力ファイル名を2回指定しています。</li> <li>⇒ 出力ファイル名の指定を1つにしてください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• コマンドラインがコンパイルオプション-oで終了しています。</li> <li>⇒ コンパイルオプション-oの後に出力ファイル名を指定してください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• インクルードファイルのパスを指定するコンパイルオプション-Iが制限値を越えました。</li> <li>⇒ コンパイルオプション-Iを8個以下にしてください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• コマンドラインがコンパイルオプション-Iで終了しています。</li> <li>⇒ コンパイルオプション-Iの後に出力ファイル名を指定してください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• コンパイルオプション-Dの次の文字列がマクロ名で使用できる文字タイプ(英字又は_)ではありません。不当なマクロ名定義を行っています。</li> <li>⇒ コンパイルオプション-Dの後に出力ファイル名を指定してください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• コマンドラインがコンパイルオプション-Dで終了しています。</li> <li>⇒ コンパイルオプション-Dの後に出力ファイル名を指定してください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• コンパイルオプション-Uの次の文字列がマクロ名で使用できる文字タイプ(英字又は_)ではありません。</li> <li>⇒ 正しいマクロ名定義を指定してください。</li> </ul> <hr/> <ul style="list-style-type: none"> <li>• cpp100で使用できないオプションを指定しています。</li> <li>⇒ 正しいオプションを指定してください。</li> </ul> |
| 11 | cannot open input file.         | <ul style="list-style-type: none"> <li>• 入力ファイルが見つかりません。</li> <li>⇒ 正しいファイル名を指定してください。</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 12 | cannot close input file.        | <ul style="list-style-type: none"> <li>• 入力ファイルをクローズできません。</li> <li>⇒ 入力ファイルを確認してください。</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 14 | cannot open output file.        | <ul style="list-style-type: none"> <li>• 出力ファイルがオープンできません。</li> <li>⇒ 正しいファイル名を指定してください。</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 15 | cannot close output file.       | <ul style="list-style-type: none"> <li>• 出力ファイルをクローズできません。</li> <li>⇒ ディスクの空き容量を確認してください。</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 16 | cannot write output file        | <ul style="list-style-type: none"> <li>• ファイルの書き込み中にエラーが発生しました。</li> <li>⇒ ディスクの空き容量を確認してください。</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 17 | input file name buffer overflow | <ul style="list-style-type: none"> <li>• 入力ファイル名のバッファがオーバーフローしました。ファイル名は、ディレクトリパス名を含みますので注意してください。</li> <li>⇒ ファイル名、パス名(標準ディレクトリ、コンパイルオプション-I指定)を短くしてください。</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |



表F.4 cpp100エラーメッセージ一覧表 (2/3)

| 番号 | エラーメッセージ                                           | エラー内容と対策                                                                                                                                                                                                       |
|----|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 18 | not enough memory for macro include file not found | <ul style="list-style-type: none"> <li>マクロ名、綴り文字を登録するためのメモリが足りません。</li> </ul> ⇒ スワップ領域を増やしてください。                                                                                                               |
| 21 | include file not found                             | <ul style="list-style-type: none"> <li>インクルードファイルがオープンできません。</li> </ul> ⇒ インクルードファイルは、カレントディレクトリ、コンパイルオプション4や環境変数で指定したディレクトリに存在します。これらのディレクトリを確認してください。                                                        |
| 22 | illegal file name error                            | <ul style="list-style-type: none"> <li>ファイル名の指定が誤っています。</li> </ul> ⇒ ファイル名の指定を正しく行ってください。                                                                                                                      |
| 23 | include file nesting over                          | <ul style="list-style-type: none"> <li>インクルードファイルのネスティングの上限(40)を越えました。</li> </ul> ⇒ インクルードファイルのネスティングを40までにしてください。                                                                                             |
| 25 | illegal identifier                                 | <ul style="list-style-type: none"> <li>#defineの記述に誤りがあります。</li> </ul> ⇒ ソースファイルを正しく記述してください。                                                                                                                   |
| 26 | illegal operation                                  | <ul style="list-style-type: none"> <li>プリプロセスコマンド#if~#elseif~#assertの演算式中に誤りがあります。</li> </ul> ⇒ 演算式を正しく記述してください。                                                                                               |
| 27 | macro argument error                               | <ul style="list-style-type: none"> <li>マクロ展開時のマクロ引数の数に誤りがあります。</li> </ul> ⇒ マクロ定義及び参照を確認して正しく記述してください。                                                                                                         |
| 28 | input buffer over flow                             | <ul style="list-style-type: none"> <li>ソースファイルリード中に入力バッファがオーバーフローしました。又は、マクロ変換中にバッファがオーバーフローしました。</li> </ul> ⇒ ソースファイルの1行を1023文字以下にしてください。マクロ変換を予想される場合は、変換結果が1023文字以下になるように変更してください。                          |
| 29 | EOF in comment                                     | <ul style="list-style-type: none"> <li>コメント中にファイルが終了しています。</li> </ul> ⇒ ソースファイルを正しく記述してください。                                                                                                                   |
| 31 | EOF in preprocess command                          | <ul style="list-style-type: none"> <li>プリプロセスコマンド中にファイルが終了しています。</li> </ul> ⇒ ソースファイルを正しく記述してください。                                                                                                             |
| 32 | unknown preprocess command                         | <ul style="list-style-type: none"> <li>不当なプリプロセスコマンドを使用しています。</li> </ul> ⇒ cpp100で使用できるプリプロセスコマンドは、次のコマンドのみです。<br>#include、#define、#undef、#if、#ifdef、#ifndef、#else、#endif、#elseif、#line、#assert、#pragma、#error |
| 33 | new_line in string                                 | <ul style="list-style-type: none"> <li>文字定数又は、文字列定数中に改行が含まれています。</li> </ul> ⇒ プログラムを正しく記述しなおしてください。                                                                                                            |
| 34 | string literal out of range 509 characters         | <ul style="list-style-type: none"> <li>文字列が509文字をこえました。</li> </ul> ⇒ 文字列は509文字以下にしてください。                                                                                                                       |
| 35 | macro replace nesting over                         | <ul style="list-style-type: none"> <li>マクロのネスティングが制限値(20)を越えました。</li> </ul> ⇒ 制限値を越えないようにしてください。                                                                                                               |
| 41 | include file error                                 | <ul style="list-style-type: none"> <li>#include命令の記述に誤りがあります。</li> </ul> ⇒ インクルードファイルを正しく記述してください。                                                                                                             |
| 43 | illegal id name                                    | <ul style="list-style-type: none"> <li>#defineコマンドの以下のマクロ名又は引数の記述に誤りがあります。<br/>           __FILE__、__LINE__、__DATE__、__TIME__</li> </ul> ⇒ ソースファイルを正しく記述してください。                                                |

表F.5 cpp100エラーメッセージ一覧表 (3/3)

| 番号 | エラーメッセージ                             | エラー内容と対策                                                                                                                                           |
|----|--------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| 44 | token buffer over flow               | <ul style="list-style-type: none"> <li>• #defineの綴り文字のバッファがオーバーフローしました。</li> </ul> ⇒ 綴り文字を短くしてください。                                                |
| 45 | illegal undef command usage          | <ul style="list-style-type: none"> <li>• #undefの記述に誤りがあります。</li> </ul> ⇒ ソースファイルを正しく記述してください。                                                      |
| 46 | undef id not found                   | <ul style="list-style-type: none"> <li>• #undefで未定義にしようとしている以下のマクロ名が定義されていません。<br/>__FILE__、__LINE__、__DATE__、__TIME__</li> </ul> ⇒ マクロ名を確認してください。 |
| 52 | illegal ifdef / ifndef command usage | <ul style="list-style-type: none"> <li>• #ifdefの記述に誤りがあります。</li> </ul> ⇒ ソースファイルを正しく記述してください。                                                      |
| 53 | elseif / else sequence erro          | <ul style="list-style-type: none"> <li>• #if~#ifdef~#ifndefがないのに#elseif又は#elseを使用しています。</li> </ul> ⇒ #elseif又は#elseは#if~#ifdef~#ifndefの後に使ってください。  |
| 54 | endif not exist                      | <ul style="list-style-type: none"> <li>• #if~#ifdef~#ifndefに対応した#endifがありません。</li> </ul> ⇒ #endifをソースファイル中に記述してください。                               |
| 55 | endif sequence error                 | <ul style="list-style-type: none"> <li>• #if~#ifdef~#ifndefがないのに#endifを使用しています。</li> </ul> ⇒ #endifは#if~#ifdef~#ifndefの後に使ってください。                  |
| 61 | illegal line command usage           | <ul style="list-style-type: none"> <li>• #lineの記述に誤りがあります。</li> </ul> ⇒ ソースファイルを正しく記述してください。                                                       |

## F.4 cpp100 ワーニングメッセージ

【表F.6】にプリプロセッサcpp100が出力するワーニングメッセージとその内容及び対処方法を示します。

表F.6 cpp100ワーニングメッセージ一覧表

| 番号 | ワーニングメッセージ                                           | エラー内容と対策                                                                                                                                              |
|----|------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| 81 | reserved id used                                     | <ul style="list-style-type: none"> <li>• cpp100が予約済みの以下のマクロ名を定義又は未定義にしようとしています。<br/>__FILE__、__LINE__、__DATE__、__TIME__</li> </ul> ⇒ 他のマクロ名を使用してください。 |
| 82 | assertion warning                                    | <ul style="list-style-type: none"> <li>• #assertの演算式の結果が0になりました。</li> </ul> ⇒ 演算式を確認してください。                                                           |
| 83 | garbage argument                                     | <ul style="list-style-type: none"> <li>• プリプロセスコマンドの後にコメント以外の文字があります。</li> </ul> ⇒ プリプロセスコマンドの後の文字はコメント(/*. . . . */)で記述してください。                       |
| 84 | escape sequence out of range for character           | <ul style="list-style-type: none"> <li>• 文字定数、文字列定数に含まれるエスケープ文字が255越えました。</li> </ul> ⇒ エスケープ文字は255までにしてください。                                          |
| 85 | redefined                                            | <ul style="list-style-type: none"> <li>• 一度定義したマクロを以前定義したときと異なる内容で再度定義しています。</li> </ul> ⇒ 以前定義した内容と比較して確認してください。                                      |
| 87 | /* within comment                                    | <ul style="list-style-type: none"> <li>• コメント内に/*を記述しています。</li> </ul> ⇒ ネストしないようにコメントを記述してください。                                                       |
| 88 | Environment variable 'NCKIN' must be 'SJIS' or 'EUC' | <ul style="list-style-type: none"> <li>• 環境変数 NCKIN に誤りがあります。</li> </ul> ⇒ NCKIN には、"SJIS"、"EUC" のいずれかを設定してください。                                      |
| 90 | 'マクロ名' in #if is not defined、 so it's treated as 0   | <ul style="list-style-type: none"> <li>• #if 文で未定義のマクロが使用されています。</li> </ul> ⇒ マクロ定義を確認してください。                                                         |

## F.5 ccom100 エラーメッセージ

【表F.7】～【表F.18】にコンパイラccom100が出力するエラーメッセージとその内容及び対処方法を示します。

表F.7 ccom100エラーメッセージ一覧表 (1/12)

| エラーメッセージ                                               | エラー内容と対策                                                                                                                                                                              |
|--------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| #pragma プラグマ名 関数名 redefined                            | <ul style="list-style-type: none"> <li>• #pragma プラグマ名 において同じ関数を重複して定義しています。</li> </ul> ⇒ #pragma プラグマ名の宣言を1回にしてください。                                                                 |
| #pragma プラグマ名 function argument is long-long or double | <ul style="list-style-type: none"> <li>• #pragma プラグマ名 で指定した関数の引数に、long long型、double型が使用されています。</li> </ul> ⇒ "#pragma プラグマ名 関数名"で指定した関数には、long long型、およびdouble型を指定できません。別の型を使用してください。 |
| #pragma プラグマ名 & function prototype mismatched          | <ul style="list-style-type: none"> <li>• #pragma プラグマ名 で指定した関数とプロトタイプ宣言の引数の内容が異なります。</li> </ul> ⇒ プロトタイプ宣言の引数と合わせてください。                                                               |
| #pragma プラグマ名 function argument is struct or union     | <ul style="list-style-type: none"> <li>• #pragma プラグマ名 で指定した関数のプロトタイプ宣言で struct / union型 を指定しています。</li> </ul> ⇒ プロトタイプ宣言でint、short型又は、サイズが2バイトのポインタ型、列挙型を指定してください。                    |
| #pragma プラグマ名 must be declared before use              | <ul style="list-style-type: none"> <li>• #pragma プラグマ名 で指定した関数の定義が、その関数の呼び出しの後に記述されています。</li> </ul> ⇒ 関数の呼び出しを行う前に宣言してください。                                                           |
| #pragma BITADDRESS variable is not _Bool type          | <ul style="list-style-type: none"> <li>• #pragma BITADDRESS で指定された変数が、_Bool型ではありません。</li> </ul> ⇒ #pragma BITADDRESS に指定する変数は、_Bool型にしてください。                                          |
| #pragma INTCALL function's argument on stack           | <ul style="list-style-type: none"> <li>• #pragma INTCALL で宣言した関数の実体をC言語で記述した場合に引き数がスタック渡しになっています。</li> </ul> ⇒ #pragma INTCALL で宣言した関数の実体をC言語で記述する場合は引き数にはレジスタ渡しとなる型を指定してください。        |
| #pragma PARAMETER functions register not allocated     | <ul style="list-style-type: none"> <li>• #pragmaPARAMETERで指定した関数で指定したレジスタは、記述できません。</li> </ul> ⇒ レジスタを正しく記述してください。                                                                    |
| 'const' is duplicate                                   | <ul style="list-style-type: none"> <li>• constを2回以上記述しています。</li> </ul> ⇒ 型修飾子を正しく記述してください。                                                                                            |
| 'far' & 'near' conflict                                | <ul style="list-style-type: none"> <li>• 同じ変数(関数)に対して near / farの宣言が一致していません。</li> </ul> ⇒ near / far を正しく記述してください。                                                                   |
| 'far' is duplicate                                     | <ul style="list-style-type: none"> <li>• farを2回以上記述しています。</li> </ul> ⇒ farを正しく記述してください。                                                                                               |
| 'near' is duplicate                                    | <ul style="list-style-type: none"> <li>• near を2回以上記述しています。</li> </ul> ⇒ nearを正しく記述してください。                                                                                            |
| 'static' is illegal storage class for argument         | <ul style="list-style-type: none"> <li>• 引数の宣言において不適当な記憶域クラスを使用しています。</li> </ul> ⇒ 正しい記憶域クラスを使用してください。                                                                                |
| 'volatile' is duplicate                                | <ul style="list-style-type: none"> <li>• volatileを2回以上記述しています。</li> </ul> ⇒ 型修飾子を正しく記述してください。                                                                                         |

表F.8 ccom100エラーメッセージ一覧表 (2/12)

| エラーメッセージ                                                      | エラー内容と対策                                                                                                                             |
|---------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| (can't read C source from filename line 行数 for error message) | <ul style="list-style-type: none"> <li>エラーが発生したソースラインを表示できません。filenameで示されるファイルがないか、行番号が、ファイルに存在しません。</li> </ul> ⇒ ファイルの存在を確認してください。 |
| (can't open C source filename for error message)              | <ul style="list-style-type: none"> <li>エラーが発生したソースファイルがオープンできません。</li> </ul> ⇒ ファイルの存在を確認してください。                                     |
| argument type given both places                               | <ul style="list-style-type: none"> <li>関数定義中の引数の宣言において、引数リストと重複して引数の宣言を行っています。</li> </ul> ⇒ 引数リストか、引数の宣言のどちらかで引数の宣言を行ってください。         |
| array of functions declared                                   | <ul style="list-style-type: none"> <li>配列宣言において関数のポインタ配列ではなく関数自身の配列を宣言しています。</li> </ul> ⇒ 関数のポインタ配列等に変更してください。                       |
| array size is not constant integer                            | <ul style="list-style-type: none"> <li>配列の宣言において要素数が定数ではありません。</li> </ul> ⇒ 要素数を定数で記述してください。                                         |
| asm()'s string must have only 1 \$b                           | <ul style="list-style-type: none"> <li>asmステートメントで \$bは一度しか記述することはできません。</li> </ul> ⇒ \$bの記述を1回にしてください。                              |
| asm()'s string must not have more than 3 \$\$ or \$\$@        | <ul style="list-style-type: none"> <li>asmステートメントで \$\$または \$\$@を3回以上記述しています。</li> </ul> ⇒ \$\$(\$\$@)の記述を2回にしてください。                 |
| auto variable's size is zero                                  | <ul style="list-style-type: none"> <li>auto領域に要素数が0の配列、あるいは要素数のない配列を宣言しています。</li> </ul> ⇒ 正しく宣言してください。                               |
| bitfield width exceeded                                       | <ul style="list-style-type: none"> <li>ビットフィールドの幅が、データ型のビット幅を超えています。</li> </ul> ⇒ ビットフィールドで宣言したデータ型のビット幅以内で記述してください。                 |
| bitfield width is not constant integer                        | <ul style="list-style-type: none"> <li>ビットフィールドのビット幅が定数ではありません。</li> </ul> ⇒ ビット幅を定数で記述してください。                                       |
| can't get bitfield address by '&' operator                    | <ul style="list-style-type: none"> <li>ビットフィールドタイプに&amp;演算子を記述しています。</li> </ul> ⇒ ビットフィールドタイプに&演算子を記述しないでください。                       |
| can't get inline function's address by '&' operator           | <ul style="list-style-type: none"> <li>インライン関数に&amp;演算子を記述しています。</li> </ul> ⇒ インライン関数に&演算子を記述しないでください。                               |
| can't get size of bitfield                                    | <ul style="list-style-type: none"> <li>ビットフィールドのサイズを取得しようとしています。</li> </ul> ⇒ ビットフィールドのサイズを取得することはできません。                             |
| can't get void value                                          | <ul style="list-style-type: none"> <li>代入式の右辺がvoid型等のように、void型のデータを取り出そうとしています。</li> </ul> ⇒ データの型を確認してください。                         |
| can't output to ファイル名                                         | <ul style="list-style-type: none"> <li>ファイルに書き込みができません。</li> </ul> ⇒ ディスクの残り容量又はファイルのパーミッションを確認してください。                               |
| can't open ファイル名                                              | <ul style="list-style-type: none"> <li>ファイルがオープンできません。</li> </ul> ⇒ ファイルのパーミッションを確認してください。                                           |
| can't refer to the range outside of the stack frame.          | <ul style="list-style-type: none"> <li>スタックフレーム領域外を参照しています。</li> </ul> ⇒ 正しく指定してください。                                                |

表F.9 ccom100エラーメッセージ一覧表 (3/12)

| エラーメッセージ                                             | エラー内容と対策                                                                                                                                       |
|------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| can't set argument                                   | <ul style="list-style-type: none"> <li>プロトタイプ宣言と実引数との型の不一致により、レジスタ(引数)に実引数をセットできません。</li> </ul> ⇒ 型の不一致を修正してください。                              |
| case value is duplicated                             | <ul style="list-style-type: none"> <li>caseの値を重複して使用しています。</li> </ul> ⇒ 1つのswitch文で同じcaseの値を使用しないでください。                                        |
| conflict declare of 変数名                              | <ul style="list-style-type: none"> <li>1度目と2度目で記憶域クラスの異なる重複定義を行っています。</li> </ul> ⇒ 変数を2度宣言する場合は、同じ記憶域クラスで行ってください。                              |
| conflict function argument type of 変数名               | <ul style="list-style-type: none"> <li>引数リストに同じ変数名があります。</li> </ul> ⇒ 変数名を変更してください。                                                            |
| declared register parameter function's body declared | <ul style="list-style-type: none"> <li>#pragma PARAMETER で宣言した関数をC言語で実体の定義を行っています。</li> </ul> ⇒ #pragma PARAMETER で宣言した関数はC言語での実体記述を行わないでください。 |
| default function argument conflict                   | <ul style="list-style-type: none"> <li>プロトタイプ宣言において、引数のデフォルト値を2回以上宣言しています。</li> </ul> ⇒ 引数のデフォルト値は、1回だけ宣言してください。                               |
| default: is duplicated                               | <ul style="list-style-type: none"> <li>defaultの値を重複して使用しています。</li> </ul> ⇒ 1つのswitch文でdefaultは1つにしてください。                                       |
| do while( struct/union ) statement                   | <ul style="list-style-type: none"> <li>do while文の式にstruct/union型を使用しています。</li> </ul> ⇒ do while文の式は、スカラ型を記述してください。                             |
| do while( void ) statement                           | <ul style="list-style-type: none"> <li>do while文の式にvoid型を使用しています。</li> </ul> ⇒ do while文の式は、スカラ型を記述してください。                                     |
| duplicate frame position defin 変数名                   | <ul style="list-style-type: none"> <li>同じ識別子を持つauto変数が2回以上記述されています。</li> </ul> ⇒ 正しく記述してください。                                                  |
| Empty declare                                        | <ul style="list-style-type: none"> <li>記憶域クラス指定子、型指定子しかありません。</li> </ul> ⇒ 宣言子を記述してください。                                                       |
| float and double not have sign                       | <ul style="list-style-type: none"> <li>floatやdoubleにsigned/unsignedを記述しています。</li> </ul> ⇒ 型指定子を正しく記述してください。                                    |
| floating point value overflow                        | <ul style="list-style-type: none"> <li>浮動小数点の即値が表現できる範囲を超えています。</li> </ul> ⇒ 範囲以内の値にしてください。                                                    |
| floating type's bitfield                             | <ul style="list-style-type: none"> <li>不当な型のビットフィールドを宣言しています。</li> </ul> ⇒ ビットフィールドは、整数型を使用してください。                                             |
| for( ; struct/union; ) statement                     | <ul style="list-style-type: none"> <li>for文の2番目の式にstruct/union型を使用しています。</li> </ul> ⇒ for文の2番目の式は、スカラ型を記述してください。                               |
| for( ; void ; ) statement                            | <ul style="list-style-type: none"> <li>for文の2番目の式にvoid型を使用しています。</li> </ul> ⇒ for文の2番目の式は、スカラ型を記述してください。                                       |
| function initialized                                 | <ul style="list-style-type: none"> <li>関数の宣言に対して初期化式を記述しています。</li> </ul> ⇒ 初期化式を削除してください。                                                      |
| function member declared                             | <ul style="list-style-type: none"> <li>構造体、共用体のメンバで関数型を指定しています。</li> </ul> ⇒ メンバを正しく記述してください。                                                  |
| function returning a function declared               | <ul style="list-style-type: none"> <li>関数の宣言においてリターン値の型が関数型になっています。</li> </ul> ⇒ 戻り値の型を関数へのポインタ型等に変更してください。                                    |

表F.10 ccom100エラーメッセージ一覧表 (4/12)

| エラーメッセージ                                                | エラー内容と対策                                                                                                                    |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| function returning an array                             | <ul style="list-style-type: none"> <li>関数の宣言においてリターン値の型が酒記列型になっています。</li> </ul> ⇒ 戻り値の型を関数へのポインタ型等に変更してください。                |
| handler function called                                 | <ul style="list-style-type: none"> <li>#pragma HANDLERで指定した関数を呼び出しています。</li> </ul> ⇒ ハンドラ関数は呼び出さないようにしてください。                |
| identifier (変数名) is duplicated                          | <ul style="list-style-type: none"> <li>変数が重複して定義されています。</li> </ul> ⇒ 変数の定義を正しく指定してください。                                     |
| if( struct/union ) statement                            | <ul style="list-style-type: none"> <li>if文の式にstruct/union型を使用しています。</li> </ul> ⇒ if文の式は、スカラ型を記述してください。                      |
| if( void ) statement                                    | <ul style="list-style-type: none"> <li>if文の式にvoid型を使用しています。</li> </ul> ⇒ if文の式は、スカラ型を記述してください。                              |
| illegal storage class for argument, 'inline' ignored    | <ul style="list-style-type: none"> <li>関数内での宣言文においてインライン関数を宣言しています。</li> </ul> ⇒ 関数外で宣言してください。                              |
| illegal storage class for argument, 'interrupt' ignored | <ul style="list-style-type: none"> <li>関数内での宣言文において割り込み関数を宣言しています。</li> </ul> ⇒ 関数外で宣言してください。                               |
| incomplete array access                                 | <ul style="list-style-type: none"> <li>不完全型の多次元配列を参照しています。</li> </ul> ⇒ 多次元配列のサイズを明記してください。                                 |
| incomplete return type                                  | <ul style="list-style-type: none"> <li>不完全な型を戻り値に記述しています。</li> </ul> ⇒ 戻り値を確認してください。                                        |
| incomplete struct get by [ ]                            | <ul style="list-style-type: none"> <li>メンバが確定していない(不完全な)構造体、共用体の配列を参照又は初期化しています。</li> </ul> ⇒ 完全な構造体、共用体を先に定義してください。        |
| incomplete struct member                                | <ul style="list-style-type: none"> <li>不完全な構造体をメンバとして記述しています。</li> </ul> ⇒ 完全な構造体を記述してください。                                 |
| incomplete struct initialized                           | <ul style="list-style-type: none"> <li>メンバが確定していない(不完全な)構造体、共用体を初期化しています。</li> </ul> ⇒ 完全な構造体、共用体を先に定義してください。               |
| incomplete struct return function call                  | <ul style="list-style-type: none"> <li>メンバが確定していない(不完全な)構造体、共用体の型をリターン値にもつ、関数を呼び出しています。</li> </ul> ⇒ 完全な構造体、共用体を先に定義してください。 |
| incomplete struct / union's member access               | <ul style="list-style-type: none"> <li>メンバが確定していない(不完全な)構造体、共用体のメンバを参照しています。</li> </ul> ⇒ 完全な構造体、共用体を先に定義してください。            |
| incomplete struct / union(タグ名)s member access           | <ul style="list-style-type: none"> <li>メンバが確定していない(不完全な)構造体、共用体のメンバを参照しています。</li> </ul> ⇒ 完全な構造体、共用体を先に定義してください。            |
| inline function have invalid argument or return code    | <ul style="list-style-type: none"> <li>インライン関数に、不正な引き数または、不正な戻り値があります。</li> </ul> ⇒ 正しい、引き数、戻り値を指定してください。                   |
| inline function is called as normal function before     | <ul style="list-style-type: none"> <li>インライン関数が通常の間数として、宣言前に呼び出されています。</li> </ul> ⇒ 関数を確認してください。                            |
| inline function's address used                          | <ul style="list-style-type: none"> <li>インライン関数のアドレスを参照しています。</li> </ul> ⇒ インライン関数のアドレスは使用しないでください。                          |



表F.11 ccom100エラーメッセージ一覧表 (5/12)

| エラーメッセージ                                          | エラー内容と対策                                                                                                                                                |
|---------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| inline function's body is not declared previously | <ul style="list-style-type: none"> <li>インライン関数の実体定義がありません。</li> </ul> ⇒ インライン関数を使用する際には、関数を呼び出すよりも前に関数の実体を定義してください。                                     |
| inline function (関数名) is recursion                | <ul style="list-style-type: none"> <li>インライン関数の再帰呼び出しはできません。</li> </ul> ⇒ 再帰呼び出しをしない様に記述してください。                                                         |
| interrupt function called                         | <ul style="list-style-type: none"> <li>#pragma INTERRUPTで指定した関数を呼び出しています。</li> </ul> ⇒ 割り込み処理関数は呼び出さないようにしてください。                                        |
| invalid environment variable : 環境変数名              | <ul style="list-style-type: none"> <li>環境変数NCKIN/NCKOUTで指定された変数名がSJIS、EUC以外が指定されています。</li> </ul> ⇒ 環境変数を確認してください。                                       |
| invalid function default argument                 | <ul style="list-style-type: none"> <li>関数のデフォルト引数が正しくありません。</li> </ul> ⇒ デフォルト引数を持つ関数のプロトタイプ宣言と、関数定義部で引数の整合が取れていない場合等に発生します。整合が取れるように記述してください。          |
| invalid push                                      | <ul style="list-style-type: none"> <li>関数引数等で void型を pushしています。</li> </ul> ⇒ voidを pushすることはできません。                                                      |
| invalid '?:' operand                              | <ul style="list-style-type: none"> <li>?: 演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の各式を確認してください。また、:の左右の式の型は、互換型である必要があります。                                      |
| invalid '!=' operands                             | <ul style="list-style-type: none"> <li>!=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                                             |
| invalid '&&' operands                             | <ul style="list-style-type: none"> <li>&amp;&amp;演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                                     |
| invalid '&' operands                              | <ul style="list-style-type: none"> <li>&amp;演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の右辺式を確認してください。                                                              |
| invalid '&=' operands                             | <ul style="list-style-type: none"> <li>&amp;=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                                         |
| invalid '()' operand                              | <ul style="list-style-type: none"> <li>()の左辺式が関数ではありません。</li> </ul> ⇒ ()の左辺式は関数又は関数へのポインタを記述してください。                                                     |
| invalid '*' operands                              | <ul style="list-style-type: none"> <li>乗算の場合*演算子の記述に誤りがあります。ポインタ演算子の場合、右辺式がポインタ型ではありません。</li> </ul> ⇒ 乗算の場合、演算子の左辺式、右辺式を確認してください。ポインタの場合、右辺の型を確認してください。 |
| invalid '*=' operands                             | <ul style="list-style-type: none"> <li>*=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                                             |
| invalid '+' operands                              | <ul style="list-style-type: none"> <li>+演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                                              |
| invalid '+=' operands                             | <ul style="list-style-type: none"> <li>+=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                                             |
| invalid '-' operands                              | <ul style="list-style-type: none"> <li>-演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                                              |
| invalid '-=' operands                             | <ul style="list-style-type: none"> <li>-=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                                             |



表F.12 ccom100エラーメッセージ一覧表 (6/12)

| エラーメッセージ                                         | エラー内容と対策                                                                                                                      |
|--------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| invalid '/=' operands                            | <ul style="list-style-type: none"> <li>• /=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                 |
| invalid '<<' operands                            | <ul style="list-style-type: none"> <li>• &lt;&lt;演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                           |
| invalid '<<=' operands                           | <ul style="list-style-type: none"> <li>• &lt;&lt;=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                          |
| invalid '<=' operands                            | <ul style="list-style-type: none"> <li>• &lt;=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                              |
| invalid '=' operand                              | <ul style="list-style-type: none"> <li>• =演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                  |
| invalid '==' operands                            | <ul style="list-style-type: none"> <li>• ==演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                 |
| invalid '>=' operands                            | <ul style="list-style-type: none"> <li>• &gt;=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                              |
| invalid '>>' operands                            | <ul style="list-style-type: none"> <li>• &gt;&gt;演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                           |
| invalid '>>=' operands                           | <ul style="list-style-type: none"> <li>• &gt;&gt;=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                          |
| invalid '[' operands                             | <ul style="list-style-type: none"> <li>• [] の左辺式が配列、ポインタ型ではありません。</li> </ul> ⇒ [] の左辺式は配列、又はポインタ型を記述してください。                   |
| invalid '^=' operands                            | <ul style="list-style-type: none"> <li>• ^=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                 |
| invalid ' =' operands                            | <ul style="list-style-type: none"> <li>•  =演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                 |
| invalid '  ' operands                            | <ul style="list-style-type: none"> <li>•    演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                |
| invalid '%=' operands                            | <ul style="list-style-type: none"> <li>• %=演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の左辺式、右辺式を確認してください。                                 |
| invalid ++ operands                              | <ul style="list-style-type: none"> <li>• ++単項演算子又は後置演算子の記述に誤りがあります。</li> </ul> ⇒ 単項演算子の場合、右辺式を確認してください。後置演算子の場合、左辺式を確認してください。 |
| invalid -- operands                              | <ul style="list-style-type: none"> <li>• --単項演算子又は後置演算子の記述に誤りがあります。</li> </ul> ⇒ 単項演算子の場合、右辺式を確認してください。後置演算子の場合、左辺式を確認してください。 |
| invalid -> used                                  | <ul style="list-style-type: none"> <li>• -&gt;の左辺式が、構造体,共用体型ではありません。</li> </ul> ⇒ ->の左辺式を、構造体,共用体型で記述してください。                  |
| invalid (? :)'s condition                        | <ul style="list-style-type: none"> <li>• 三項演算子の記述に誤りがあります。</li> </ul> ⇒ 三項演算式を確認してください。                                       |
| Invalid array type                               | <ul style="list-style-type: none"> <li>• 不完全型の配列を宣言することはできません。</li> </ul> ⇒ 多次元配列の宣言時、配列の要素数を明示してください。                        |
| Invalid operation for pointer to incomplete type | <ul style="list-style-type: none"> <li>• 不完全型へのポインタに対して無効な演算をしています。</li> </ul> ⇒ 構造体のメンバを定義する、または配列の要素数を明示して完全型にしてください。       |

表F.13 ccom100エラーメッセージ一覧表 (7/12)

| エラーメッセージ                                                   | エラー内容と対策                                                                                                                                |
|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Invalid #pragma OS拡張機能 interrupt number                    | <ul style="list-style-type: none"> <li>• #pragma OS拡張機能 で記述したINT番号は、指定することができません。</li> </ul> ⇒ 正しく指定してください。                             |
| Invalid #pragma INTCALL interrupt number                   | <ul style="list-style-type: none"> <li>• #pragma INTCALL で記述したINT番号は、指定することができません。</li> </ul> ⇒ 正しく指定してください。                            |
| Invalid #pragma SPECIAL special page number (NC30、NC308のみ) | <ul style="list-style-type: none"> <li>• #pragma SPECIAL で記述した番号またはフォーマット指定が間違っています。</li> </ul> ⇒ 正しく指定してください。                          |
| invalid #pragma INTERRUPT vector number                    | <ul style="list-style-type: none"> <li>• #pragma INTERRUPT で記述した番号またはフォーマット指定が間違っています。</li> </ul> ⇒ 正しく指定してください。                        |
| invalid CAST operand                                       | <ul style="list-style-type: none"> <li>• cast演算子に誤りがあります。void型を他の型にキャスト及び、構造体、共用体からもしくは他の構造体、共用体へのキャストはできません。</li> </ul> ⇒ 正しく記述してください。 |
| invalid asm()'s argument                                   | <ul style="list-style-type: none"> <li>• asmステートメントに使用できる変数は、auto変数と引数です。</li> </ul> ⇒ auto変数か引数で記述してください。                              |
| invalid bitfield declare                                   | <ul style="list-style-type: none"> <li>• ビットフィールドの宣言で誤りがあります。</li> </ul> ⇒ 正しく記述してください。                                                 |
| invalid break statements                                   | <ul style="list-style-type: none"> <li>• break文を記述できないところで使用しています。</li> </ul> ⇒ witch、while、do-while、forのなかで記述してください。                   |
| invalid case statements                                    | <ul style="list-style-type: none"> <li>• switch文に誤りがあります。</li> </ul> ⇒ switch文を正しく記述してください。                                             |
| invalid case value                                         | <ul style="list-style-type: none"> <li>• caseの値に誤りがあります。</li> </ul> ⇒ 整数型、列挙型の定数を記述してください。                                              |
| invalid cast operator                                      | <ul style="list-style-type: none"> <li>• cast演算子の記述に誤りがあります。</li> </ul> ⇒ 正しく記述してください。                                                  |
| invalid continue statements                                | <ul style="list-style-type: none"> <li>• continue文を記述できないところで使用しています。</li> </ul> ⇒ while、do-while、forのなかで記述してください。                      |
| invalid default statements                                 | <ul style="list-style-type: none"> <li>• switch文に誤りがあります。</li> </ul> ⇒ switch文を正しく記述してください。                                             |
| invalid enumerator initialized                             | <ul style="list-style-type: none"> <li>• 列挙子の初期値に変数名を記述するなど誤った指定を行っています。</li> </ul> ⇒ 列挙子の初期値を正しく記述してください。                              |
| invalid function argument                                  | <ul style="list-style-type: none"> <li>• 関数定義中の引数の宣言において、引数リストに含まれない引数を宣言しています。</li> </ul> ⇒ 引数リストに存在する変数を宣言してください。                     |
| invalid function's argument declaration                    | <ul style="list-style-type: none"> <li>• 関数の引数の宣言に誤りがあります。</li> </ul> ⇒ 正しく記述してください。                                                    |
| invalid function declare                                   | <ul style="list-style-type: none"> <li>• 関数定義に誤りがあります。</li> </ul> ⇒ エラーが発生した行か、その直前の関数定義を確認してください。                                      |

表F.14 ccom100エラーメッセージ一覧表 (8/12)

| エラーメッセージ                             | エラー内容と対策                                                                                                                                          |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| invalid initializer                  | <ul style="list-style-type: none"> <li>初期化式に誤りがあります。括弧が多過ぎる、初期化式の数が多く、関数内のstatic変数をauto変数で初期化している、変数を変数で初期化しているなど。</li> </ul> ⇒ 初期化式を正しく記述してください。 |
| invalid initializer of 変数名           | <ul style="list-style-type: none"> <li>初期化式に誤りがあります。ビットフィールドの初期化式に対して変数を記述しているなど。</li> </ul> ⇒ 初期化式を正しく記述してください。                                  |
| invalid initializer on array         | <ul style="list-style-type: none"> <li>初期化式に誤りがあります。</li> </ul> ⇒ 括弧内の初期化式の数が、配列要素の数、構造体メンバの数と一致しているかを確認してください。                                   |
| invalid initializer on char array    | <ul style="list-style-type: none"> <li>初期化式に誤りがあります。</li> </ul> ⇒ 括弧内の初期化式の数が、配列要素の数、構造体メンバの数と一致しているかを確認してください。                                   |
| invalid initializer on scalar        | <ul style="list-style-type: none"> <li>初期化式に誤りがあります。</li> </ul> ⇒ 括弧内の初期化式の数が、配列要素の数、構造体メンバの数と一致しているかを確認してください。                                   |
| invalid initializer on struct        | <ul style="list-style-type: none"> <li>初期化式に誤りがあります。</li> </ul> ⇒ 括弧内の初期化式の数が、配列要素の数、構造体メンバの数と一致しているかを確認してください。                                   |
| invalid initializer, too many brace  | <ul style="list-style-type: none"> <li>auto記憶域クラスのスカラ型の初期化式において括弧{}が多過ぎます。</li> </ul> ⇒ 括弧{}の数を減らしてください。                                          |
| invalid lvalue                       | <ul style="list-style-type: none"> <li>代入文の左辺が、lvalueではありません。</li> </ul> ⇒ 辺式に代入可能な式を記述してください。                                                    |
| invalid lvalue at '=' operator       | <ul style="list-style-type: none"> <li>代入文の左辺が、lvalueではありません。</li> </ul> ⇒ 左辺式に代入可能な式を記述してください。                                                   |
| invalid member                       | <ul style="list-style-type: none"> <li>メンバ参照の記述に誤りがあります。</li> </ul> ⇒ 正しく記述してください。                                                                |
| invalid member used                  | <ul style="list-style-type: none"> <li>メンバ参照の記述に誤りがあります。</li> </ul> ⇒ 正しく記述してください。                                                                |
| invalid redefined type name of (識別子) | <ul style="list-style-type: none"> <li>typedefで同じ識別子名を2回以上定義しています。</li> </ul> ⇒ 識別子名を正しく記述してください。                                                 |
| invalid return type                  | <ul style="list-style-type: none"> <li>関数の戻り値の型が正しくありません。</li> </ul> ⇒ 正しく記述してください。                                                               |
| invalid sign specifier               | <ul style="list-style-type: none"> <li>signed/unsignedを2回以上記述しています。</li> </ul> ⇒ 型指定子を正しく記述してください。                                                |
| invalid strage class for data        | <ul style="list-style-type: none"> <li>記憶クラスの指定に誤りがあります。</li> </ul> ⇒ 正しく記述してください。                                                                |
| invalid struct or union type         | <ul style="list-style-type: none"> <li>列挙型のデータに対して構造体、共用体のメンバを参照しています。</li> </ul> ⇒ 正しく記述してください。                                                  |
| invalid truth expression             | <ul style="list-style-type: none"> <li>条件式(?)の1つめの式でvoid, struct, union型を使用しています。</li> </ul> ⇒ スカラ型で記述してください。                                     |

表F.15 ccom100エラーメッセージ一覧表 (9/12)

| エラーメッセージ                              | エラー内容と対策                                                                                                                                     |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| invalid type specifier                | <ul style="list-style-type: none"> <li>int int i; 等のように同じ型指定子を2回以上記述しているか、float int i; 等のように矛盾した型指定子を記述しています。</li> </ul> ⇒ 型指定子を正しく記述してください。 |
| invalid type's bitfield               | <ul style="list-style-type: none"> <li>不当な型のビットフィールドを宣言しています。</li> </ul> ⇒ ビットフィールドは整数型を使用してください。                                            |
| invalid type specifier,long long long | <ul style="list-style-type: none"> <li>longを3個以上記述して型宣言しています。</li> </ul> ⇒ 型宣言を確認してください。                                                     |
| invalid unary '!' operands            | <ul style="list-style-type: none"> <li>! 単項演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の右辺式を確認してください。                                                    |
| invalid unary '+' operands            | <ul style="list-style-type: none"> <li>+単項演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の右辺式を確認してください。                                                     |
| invalid unary '-' operands            | <ul style="list-style-type: none"> <li>-単項演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の右辺式を確認してください。                                                     |
| invalid unary '~' operands            | <ul style="list-style-type: none"> <li>~単項演算子の記述に誤りがあります。</li> </ul> ⇒ 演算子の右辺式を確認してください。                                                     |
| invalid void type                     | <ul style="list-style-type: none"> <li>void型指定子にlongやsignedの型指定子を記述しています。</li> </ul> ⇒ 型指定子を正しく記述してください。                                     |
| invalid void type, int assumed        | <ul style="list-style-type: none"> <li>void型の変数は宣言できません。int型として処理を続けます。</li> </ul> ⇒ 型指定子を正しく記述してください。                                       |
| invalid size of bitfield              | <ul style="list-style-type: none"> <li>ビットフィールドのサイズを取得しようとしています。</li> </ul> ⇒ この宣言ではビットフィールドを記述しないでください。                                     |
| invalid switch statement              | <ul style="list-style-type: none"> <li>switch文の記述に誤りがあります。</li> </ul> ⇒ 正しく記述してください。                                                         |
| label ラベル redefine                    | <ul style="list-style-type: none"> <li>1つの関数内で同じラベルを2度定義しています。</li> </ul> ⇒ どちらかのラベルの名前を変更してください。                                            |
| long long type's bitfield             | <ul style="list-style-type: none"> <li>long long型のビットフィールドを記述しています。</li> </ul> ⇒ long long 型はビットフィールドに宣言できません。別の型で宣言してください。                  |
| mismatch prototyped parameter type    | <ul style="list-style-type: none"> <li>プロトタイプ宣言で宣言した時と引数の型が異なります。</li> </ul> ⇒ 引数の型を確認してください。                                                |
| No #pragma ENDASM                     | <ul style="list-style-type: none"> <li>#pragma ASM と対になる#pragma ENDASM がありません。</li> </ul> ⇒ #pragma ENDASM を記述してください。                        |
| No declarator                         | <ul style="list-style-type: none"> <li>宣言文が不完全です。</li> </ul> ⇒ 完全な宣言文を記述してください。                                                              |
| Not enough memory                     | <ul style="list-style-type: none"> <li>メモリ空間が不足しています。</li> </ul> ⇒ スワップ領域を増やしてください。                                                          |
| not have 'long char'                  | <ul style="list-style-type: none"> <li>longとcharを同時に記述しています。</li> </ul> ⇒ 型指定子を正しく記述してください。                                                  |
| not have 'long float'                 | <ul style="list-style-type: none"> <li>longとfloatを同時に記述しています。</li> </ul> ⇒ 型指定子を正しく記述してください。                                                 |
| not have 'long short'                 | <ul style="list-style-type: none"> <li>longとshortを同時に記述しています。</li> </ul> ⇒ 型指定子を正しく記述してください。                                                 |

表F.16 ccom100エラーメッセージ一覧表 (10/12)

| エラーメッセージ                                                           | エラー内容と対策                                                                                                                      |
|--------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| not static initializer for 変数名                                     | <ul style="list-style-type: none"> <li>static変数の初期化式に誤りがあります。初期化式が関数呼び出しになっているなど。</li> </ul> ⇒ 初期化式を正しく記述してください。              |
| not struct or union type                                           | <ul style="list-style-type: none"> <li>-&gt;の左辺式が、構造体,共用体型ではありません。</li> </ul> ⇒ ->の左辺式を、構造体,共用体型で記述してください。                    |
| redeclare of 変数名                                                   | <ul style="list-style-type: none"> <li>変数名が重複して定義されています。</li> </ul> ⇒ どちらかの変数名を変更してください。                                      |
| redeclare of 列挙子                                                   | <ul style="list-style-type: none"> <li>列挙子が重複して定義されています。</li> </ul> ⇒ どちらかの列挙子の名前を変更してください。                                   |
| redefine function 関数名                                              | <ul style="list-style-type: none"> <li>関数名で示される関数が2度定義されています</li> </ul> ⇒ 関数は1度しか定義できません。どちらかの関数名を変更してください。                   |
| redefinition tag of enum タグ名                                       | <ul style="list-style-type: none"> <li>列挙を二重に定義しています。</li> </ul> ⇒ 列挙の定義は1回にしてください。                                           |
| redefinition tag of struct タグ名                                     | <ul style="list-style-type: none"> <li>構造体を二重に定義しています。</li> </ul> ⇒ 構造体の定義は1回にしてください。                                         |
| redefinition tag of union タグ名                                      | <ul style="list-style-type: none"> <li>共用体を二重に定義しています。</li> </ul> ⇒ 共用体の定義は1回にしてください。                                         |
| reinitialized of 変数名                                               | <ul style="list-style-type: none"> <li>同じ変数に対して初期化式を2度指定しています。</li> </ul> ⇒ 初期化式を1つにしてください。                                   |
| reinitialized of 変数名 'restrict' is duplicate                       | <ul style="list-style-type: none"> <li>restrictの宣言が重複しています。</li> </ul> ⇒ restrictの宣言は1回にしてください。                               |
| size of incomplete array type                                      | <ul style="list-style-type: none"> <li>大きさが不明な配列のsizeofを求めています。無効なサイズです。</li> </ul> ⇒ 配列の大きさを指定してください。                        |
| size of incomplete type                                            | <ul style="list-style-type: none"> <li>sizeof演算子のオペランドに定義されていない構造体、共用体を記述しています。</li> </ul> ⇒ 構造体、共用体を先に定義してください。              |
| size of void                                                       | <ul style="list-style-type: none"> <li>voidのサイズを求めています。無効なサイズです。</li> </ul> ⇒ voidのサイズは求められません。                               |
| Sorry stack frame memory exhaust, max. 128 bytes but now nnn bytes | <ul style="list-style-type: none"> <li>スタックフレーム上に確保可能な引数は最大128バイトまでです。現在 nnnバイト使用しています。</li> </ul> ⇒ 引数のサイズあるいは引数の個数減らしてください。 |
| Sorry, compilation terminated because of these errors in関数名        | <ul style="list-style-type: none"> <li>関数名で示される関数でエラーが発生しました。コンパイルを中止します。</li> </ul> ⇒ このメッセージが出力される以前のエラーを修正してください。          |
| Sorry, compilation terminated because of too many errors.          | <ul style="list-style-type: none"> <li>ソースファイル中のエラーがエラーの上限(50個)を超えました。</li> </ul> ⇒ このメッセージが出力される以前のエラーを修正してください。             |
| struct or enum's tag used for union                                | <ul style="list-style-type: none"> <li>構造体、列挙型のタグ名を共用体のタグ名として使用しています。</li> </ul> ⇒ タグ名を変更してください。                              |

表F.17 ccom100エラーメッセージ一覧表 (11/12)

| エラーメッセージ                                        | エラー内容と対策                                                                                                                                                            |
|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| struct or union's tag used for enum             | <ul style="list-style-type: none"> <li>構造体、共用体のタグ名を列挙型のタグ名として使用しています。</li> </ul> ⇒ タグ名を変更してください。                                                                    |
| struct or union,enum does not have long or sign | <ul style="list-style-type: none"> <li>struct/union/enum型指定子にlongやsignedの型指定子を記述しています。</li> </ul> ⇒ 型指定子を正しく記述してください。                                               |
| switch's condition is floating                  | <ul style="list-style-type: none"> <li>switch文の式に浮動小数点型を使用しています。</li> </ul> ⇒ 整数型、列挙型を使用してください。                                                                     |
| switch's condition is void                      | <ul style="list-style-type: none"> <li>switch文の式にvoid型を使用しています。</li> </ul> ⇒ 整数型、列挙型を使用してください。                                                                      |
| switch's condition must integer                 | <ul style="list-style-type: none"> <li>switch文の式に整数型、列挙型以外の型を使用しています。</li> </ul> ⇒ 整数型、列挙型を使用してください。                                                                |
| syntax error                                    | <ul style="list-style-type: none"> <li>文法エラーです。</li> </ul> ⇒ 正しく記述してください。                                                                                           |
| System Error...                                 | <ul style="list-style-type: none"> <li>通常は発生しません(内部エラーです)。本エラーの発生より以前に、発生したエラーに伴い発生する場合があります。</li> </ul> ⇒ 本エラー発生以前のエラーを全て取り除いても、本エラーが発生する場合は、メッセージの内容を弊社までご連絡ください。 |
| too big data-length                             | <ul style="list-style-type: none"> <li>32ビット以上のアドレスを取得しようとしています。</li> </ul> ⇒ ご使用になれるマイコンに応じたアドレスに収まる値を設定してください。                                                    |
| too big address                                 | <ul style="list-style-type: none"> <li>32ビット以上のアドレスを設定しようとしています。</li> </ul> ⇒ ご使用になれるマイコンに応じたアドレスに収まる値を設定してください。                                                    |
| too many storage class of typedef               | <ul style="list-style-type: none"> <li>宣言中にextern/typedef/static/auto/registerなどの記憶域クラス指定子を2以上記述しています。</li> </ul> ⇒ 記憶域クラス指定子を2回以上指定しないでください。                       |
| type redeclaration of 変数名                       | <ul style="list-style-type: none"> <li>1度目と2度目で型の異なる重複定義を行っています。</li> </ul> ⇒ 変数を2度宣言する場合は同じ型で行ってください。                                                              |
| typedef initialized                             | <ul style="list-style-type: none"> <li>typedefで宣言した変数に初期化式を記述しています。</li> </ul> ⇒ 初期化式を削除してください。                                                                     |
| uncomplete array pointer operation              | <ul style="list-style-type: none"> <li>不完全型の配列に対してポインタ参照しようとしています。</li> </ul> ⇒ 完全な配列を先に定義してください。                                                                   |
| undefined label "ラベル" used                      | <ul style="list-style-type: none"> <li>gotoの分岐先のラベルが関数内に定義されていません。</li> </ul> ⇒ 関数内に分岐先のラベルを定義してください。                                                               |
| union or enum's tag used for struct             | <ul style="list-style-type: none"> <li>共用体、列挙型のタグ名を構造体のタグ名として使用しています。</li> </ul> ⇒ タグ名を変更してください。                                                                    |
| unknown function argument 変数名                   | <ul style="list-style-type: none"> <li>引数リストにない引数を指定しています。</li> </ul> ⇒ 引数を確認してください。                                                                                |
| unknown member メンバ名 used                        | <ul style="list-style-type: none"> <li>構造体,共用体のメンバに登録されていないメンバを参照しています。</li> </ul> ⇒ メンバ名を確認してください。                                                                 |

表F.18 ccom100エラーメッセージ一覧表 (12/12)

| エラーメッセージ                                               | エラー内容と対策                                                                                                                                                  |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| unknown pointer to structure idetifier "変数名"           | <ul style="list-style-type: none"> <li>• <code>-&gt;</code> の左辺式が、構造体,共用体型ではありません。</li> </ul> ⇒ <code>-&gt;</code> の左辺式を、構造体,共用体型で記述してください。               |
| unknown size of struct or union                        | <ul style="list-style-type: none"> <li>• 大きさの確定していない、不完全な構造体、共用体を使用しています。</li> </ul> ⇒ 構造体、共用体の変数を宣言する前に、構造体、共用体を宣言してください。                                |
| unknown structure idetifier "変数名"                      | <ul style="list-style-type: none"> <li>• <code>.</code> の左辺式が、構造体,共用体型ではありません。</li> </ul> ⇒ <code>.</code> の左辺式を、構造体,共用体型で記述してください。                       |
| unknown variable "変数名" used in asm()                   | <ul style="list-style-type: none"> <li>• <code>asm</code> ステートメントにおいて、未定義の変数名を使用しています。</li> </ul> ⇒ 変数を定義してください。                                          |
| unknown variable 変数名                                   | <ul style="list-style-type: none"> <li>• 未定義の変数名を使用しています。</li> </ul> ⇒ 変数を定義してください。                                                                       |
| unknown variable 変数名 used                              | <ul style="list-style-type: none"> <li>• 未定義の変数名を使用しています。</li> </ul> ⇒ 変数を定義してください。                                                                       |
| void array is invalid type, int array assumed          | <ul style="list-style-type: none"> <li>• <code>void</code> 型の配列は宣言できません。<code>int</code> 型の配列として処理を続けます。</li> </ul> ⇒ 型指定子を正しく記述してください。                   |
| void value can't return                                | <ul style="list-style-type: none"> <li>• <code>void</code> でキャストされた値を関数の戻り値に記述しています。</li> </ul> ⇒ 正しく記述してください。                                            |
| while( struct/union ) statement                        | <ul style="list-style-type: none"> <li>• <code>while</code> 文の式に <code>struct/union</code> 型を使用しています。</li> </ul> ⇒ <code>while</code> 文の式は、スカラ型を記述してください。 |
| while( void ) statement                                | <ul style="list-style-type: none"> <li>• <code>while</code> 文の式に <code>void</code> 型を使用しています。</li> </ul> ⇒ <code>while</code> 文の式は、スカラ型を記述してください。         |
| multiple #pragma EXT4MPTR's pointer, ignored( NC30 のみ) | <ul style="list-style-type: none"> <li>• <code>#pragma EXT4MPTR</code> が2個以上宣言されています。</li> </ul> ⇒ 宣言を一つにしてください。                                          |
| zero size array member                                 | <ul style="list-style-type: none"> <li>• サイズがゼロの配列です。</li> </ul> ⇒ サイズを明確にしてください。                                                                         |
|                                                        | <ul style="list-style-type: none"> <li>• 構造体のメンバにサイズがゼロの配列があります。</li> </ul> ⇒ サイズがゼロの配列を構造体のメンバにすることはできません。                                               |
| '関数名' is recursion, then inline is ignored             | <ul style="list-style-type: none"> <li>• インライン宣言された'関数名'が再帰呼び出しされています。インライン宣言を無視します。</li> </ul> ⇒ 再帰呼び出しをしない様に記述してください。                                   |



## F.6 ccom100 ワーニングメッセージ

【表F.19】～【表F.27】にコンパイラccom100が出力するワーニングメッセージとその内容及び対処方法を示します。

表F.19 ccom100ワーニングメッセージ一覧表 (1/9)

| ワーニングメッセージ                                                 | ワーニング内容と対策                                                                                                                                                      |
|------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| #pragma プラグマ名 & HANDLER both specified                     | <ul style="list-style-type: none"> <li>1つの関数に#pragma プラグマ名 と#pragma HANDLERの両方を指定しています。</li> <li>⇒ #pragma プラグマ名と#pragma HANDLER は、排他的に指定してください。</li> </ul>     |
| #pragma プラグマ名 & INTERRUPT both specified                   | <ul style="list-style-type: none"> <li>1つの関数に#pragma プラグマ名 と#pragma INTERRUPTの両方を指定しています。</li> <li>⇒ #pragma プラグマ名と#pragma INTERRUPT は、排他的に指定してください。</li> </ul> |
| #pragma プラグマ名 & TASK both specified                        | <ul style="list-style-type: none"> <li>1つの関数に#pragma プラグマ名と#pragma TASKの両方を指定しています。</li> <li>⇒ #pragma プラグマ名と#pragma TASKは、排他的に指定してください。</li> </ul>             |
| #pragma プラグマ名 format error                                 | <ul style="list-style-type: none"> <li>#pragma プラグマ名 の記述に誤りがあります。</li> <li>⇒ 正しく記述してください。</li> </ul>                                                            |
| #pragma プラグマ名 format error, ignored                        | <ul style="list-style-type: none"> <li>#pragma プラグマ名 の記述に誤りがあります。この行を無視します。</li> <li>⇒ 正しく記述してください。</li> </ul>                                                  |
| #pragma プラグマ名 not function, ignored                        | <ul style="list-style-type: none"> <li>#pragma プラグマ名 において関数でない名前を記述しています。</li> <li>⇒ 関数名で記述してください。</li> </ul>                                                   |
| #pragma プラグマ名's function must be pre-declared, ignored     | <ul style="list-style-type: none"> <li>#pragma プラグマ名で指定した関数が宣言されていません。</li> <li>⇒ #pragma プラグマ名で指定する関数は、予めプロトタイプ宣言を行ってください。</li> </ul>                          |
| #pragma プラグマ名's function must be prototyped, ignored       | <ul style="list-style-type: none"> <li>#pragma プラグマ名で指定した関数が、プロトタイプ宣言されていません。</li> <li>⇒ #pragma プラグマ名で指定する関数は、予めプロトタイプ宣言を行ってください。</li> </ul>                   |
| #pragma プラグマ名's function return type invalid, ignored      | <ul style="list-style-type: none"> <li>#pragma プラグマ名で指定された関数のリターン値の型が不当です。</li> <li>⇒ リターン値の型は、struct、union、double型以外を指定してください。</li> </ul>                      |
| #pragma プラグマ名 unknown switch, ignored                      | <ul style="list-style-type: none"> <li>#pragma プラグマ名で不正な switch を記述しています。</li> <li>⇒ 正しい switch を指定してください。</li> </ul>                                           |
| #pragma プラグマ名 variable initialized, initialization ignored | <ul style="list-style-type: none"> <li>#pragma プラグマ名 で指定した変数を初期化しています。初期化を無視します。</li> <li>⇒ #pragma プラグマ名 か、初期化式のどちらかを削除してください。</li> </ul>                      |
| #pragma ASM line too long, then cut                        | <ul style="list-style-type: none"> <li>#pragma ASM で記述できる一行の文字数1024バイトを越えています。</li> <li>⇒ 1024バイト以内で記述してください。</li> </ul>                                        |



表F.20 ccom100ワーニングメッセージ一覧表 (2/9)

| ワーニングメッセージ                                                           | ワーニング内容と対策                                                                                                                                                                                                                                                                            |
|----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| #pragma directive conflict                                           | <ul style="list-style-type: none"> <li>1つの関数に対して、異なる機能の<b>#pragma</b> を指定しています。</li> </ul> ⇒ 正しく記述してください。                                                                                                                                                                             |
| #pragma DMAC duplicate                                               | <ul style="list-style-type: none"> <li><b>#pragma DMAC</b> を2回以上定義しています。</li> </ul> ⇒ <b>#pragma DMAC</b> を正しく定義してください。                                                                                                                                                               |
| #pragma DMAC variable must be far pointer to object for 変数名, ignored | <ul style="list-style-type: none"> <li><b>#pragma DMAC</b> 宣言された変数は、オブジェクト型(または不完全型)への<b>far</b>ポインタである必要があります。<b>DMAC</b>宣言を無視します。</li> </ul> ⇒ <b>#pragma DMAC</b> を正しく定義してください。                                                                                                    |
| #pragma DMAC variable must be unsigned long for 変数名, ignored         | <ul style="list-style-type: none"> <li><b>#pragma DMAC</b>宣言された変数は、<b>unsigned long</b>型である必要があります。<b>DMAC</b>宣言を無視します。</li> </ul> ⇒ <b>#pragma DMAC</b> を正しく定義してください。                                                                                                                |
| #pragma DMAC's variable must be pre-declared, ignored                | <ul style="list-style-type: none"> <li><b>#pragma DMAC</b> 宣言された変数は、型宣言がされている必要があります。</li> </ul> ⇒ <b>#pragma DMAC</b> を正しく定義してください。                                                                                                                                                  |
| #pragma DMAC, register conflict                                      | <ul style="list-style-type: none"> <li><b>#pragma DMAC</b>宣言 において同一レジスタに複数割り当てようとしています。</li> </ul> ⇒ <b>#pragma DMAC</b> を正しく定義してください。                                                                                                                                                |
| #pragma DMAC, unknown register name used                             | <ul style="list-style-type: none"> <li><b>#pragma DMAC</b>宣言 において不明なレジスタが使用されています。</li> </ul> ⇒ <b>#pragma DMAC</b> を正しく定義してください。                                                                                                                                                     |
| #pragma JSRA illegal location, ignored                               | <ul style="list-style-type: none"> <li>関数のスコープ内に<b>#pragma JSRA</b>を記述しています。</li> </ul> ⇒ <b>#pragma JSRA</b> は、関数外に記述してください。                                                                                                                                                         |
| #pragma JSRW illegal location, ignored                               | <ul style="list-style-type: none"> <li>関数のスコープ内に<b>#pragma JSRW</b>を記述しています。</li> </ul> ⇒ <b>#pragma JSRW</b> は、関数外に記述してください。                                                                                                                                                         |
| #pragma PARAMETER function's address used                            | <ul style="list-style-type: none"> <li><b>#pragma PARAMETER</b>で指定された関数のアドレスを参照しています。</li> </ul> ⇒ 参照しないでください。                                                                                                                                                                        |
| #pragma control for function duplicate, ignored                      | <ul style="list-style-type: none"> <li><b>#pragma</b>で同じ関数に対して、<b>INTERRUPT</b>, <b>TASK</b>, <b>HANDLER</b>, <b>CYCHANDLER</b>, <b>ALMHANDLER</b> を重複して指定しています。</li> </ul> ⇒ <b>INTERRUPT</b> 、 <b>TASK</b> 、 <b>HANDLER</b> 、 <b>CYCHANDLER</b> 、 <b>ALMHANDLER</b> のうち1つを指定してください。 |
| #pragma unknown switch, ignored                                      | <ul style="list-style-type: none"> <li><b>#pragma</b> に対して不正なスイッチを指定しています。<b>#pragma</b> 宣言を無視します。</li> </ul> ⇒ 正しいスイッチを指定してください。                                                                                                                                                     |
| 'auto' is illegal storage class                                      | <ul style="list-style-type: none"> <li>不当な記憶域クラスを使用しています。</li> </ul> ⇒ 正しい記憶域クラスを指定してください。                                                                                                                                                                                            |
| 'register' is illegal storage class                                  | <ul style="list-style-type: none"> <li>不当な記憶域クラスを使用しています。</li> </ul> ⇒ 正しい記憶域クラスを指定してください。                                                                                                                                                                                            |
| argument is define by 'typedef', 'typedef' ignored                   | <ul style="list-style-type: none"> <li>引数の宣言において<b>typedef</b>を使用しています。<b>typedef</b>を無視します。</li> </ul> ⇒ <b>typedef</b> を削除してください。                                                                                                                                                   |
| assign far pointer to near pointer, bank value ignored               | <ul style="list-style-type: none"> <li><b>far</b>ポインタを<b>near</b>ポインタに代入しています。バンクアドレスを無効にします。</li> </ul> ⇒ データの型、 <b>near/far</b> を確認してください。                                                                                                                                          |

表F.21 ccom100ワーニングメッセージ一覧表 (3/9)

| ワーニングメッセージ                                               | ワーニング内容と対策                                                                                                                                                          |
|----------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| assignment from const pointer to non-const pointer       | <ul style="list-style-type: none"> <li>constポインタから非constポインタへの代入により、const性が失われます。</li> </ul> ⇒ 記述を確認してください。記述が正しい場合は、このワーニングは無視してください。                              |
| assignment from volatile pointer to non-volatile pointer | <ul style="list-style-type: none"> <li>volatileポインタから非volatileポインタへの代入により、volatile性が失われます。</li> </ul> ⇒ 記述を確認してください。記述が正しい場合は、このワーニングは無視してください。                     |
| assignment in comparison statement                       | <ul style="list-style-type: none"> <li>比較式に代入文を記述しています。</li> </ul> ⇒ "==" と記述するところを誤って "=" と記述している可能性があります。故意にそう記述したものかを確認してください。                                   |
| block level extern variable initialize forbid,ignored    | <ul style="list-style-type: none"> <li>関数内のextern変数宣言で初期化式を記述しています。</li> </ul> ⇒ 初期化式を削除するか、記憶域クラスを変更してください。                                                        |
| can't get address from register storage class variable   | <ul style="list-style-type: none"> <li>register記憶域クラスの変数に&amp;演算子を記述しています。</li> </ul> ⇒ register記憶域クラスの変数に&演算子を記述しないでください。                                          |
| can't get size of bitfield                               | <ul style="list-style-type: none"> <li>sizeof演算子のオペランドにビットフィールド型を記述しています。</li> </ul> ⇒ sizeof演算子のオペランドを正しく記述してください。                                                 |
| can't get size of function                               | <ul style="list-style-type: none"> <li>sizeof演算子のオペランドに関数名を記述しています。</li> </ul> ⇒ sizeof演算子のオペランドに関数名を記述しています。                                                       |
| can't get size of function, unit size 1 assumed          | <ul style="list-style-type: none"> <li>関数へのポインタを++,--しています。増分、減分の値を1として処理を継続します。</li> </ul> ⇒ 関数へのポインタを++, --しないでください。                                              |
| char array initialized by wchar_t string                 | <ul style="list-style-type: none"> <li>char型の初期化式をwchar_t型の文字列で初期化しています。</li> </ul> ⇒ 初期化式の型を合わせてください。                                                              |
| case value is out of range                               | <ul style="list-style-type: none"> <li>caseの値がswitchの引数の範囲を越えています。</li> </ul> ⇒ 正しく記述してください。                                                                        |
| character buffer overflow                                | <ul style="list-style-type: none"> <li>文字列のサイズが512文字を超えました。</li> </ul> ⇒ 511文字以下で記述してください。                                                                          |
| character constant too long                              | <ul style="list-style-type: none"> <li>文字定数(シングルクォートに囲まれた文字)の文字数が多すぎます。</li> </ul> ⇒ 正しく記述してください。                                                                   |
| constant variable assignment                             | <ul style="list-style-type: none"> <li>const型修飾子で指定した変数に対して代入しています。</li> </ul> ⇒ 代入先の宣言部を確認してください。                                                                  |
| cyclic or alarm handler function has argument            | <ul style="list-style-type: none"> <li>#pragma CYHANDLER又はALMHANDLERで指定した関数が、引数を使用しています。</li> </ul> ⇒ #pragma CYHANDLER又はALMHANDLERで指定した関数は、引数を使用できません。引数を削除してください。 |
| enumerator value overflow size of unsigned char          | <ul style="list-style-type: none"> <li>コンパイルオプション-fCE使用時において、列挙子の値が255を越えました。</li> </ul> ⇒ 255以下で表現できるように記述してください。                                                  |
| enumerator value overflow size of unsigned int           | <ul style="list-style-type: none"> <li>列挙子の値が65535を越えました。</li> </ul> ⇒ 65535以下で表現できるように記述してください。                                                                    |

表F.22 ccom100ワーニングメッセージ一覧表 (4/9)

| ワーニングメッセージ                                            | ワーニング内容と対策                                                                                                                                                            |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enum's bitfield                                       | <ul style="list-style-type: none"> <li>ビットフィールドのメンバに列挙型を用いて定義しています。</li> </ul> ⇒ 違う型のメンバを用いてください。                                                                     |
| external variable initialized,change to public        | <ul style="list-style-type: none"> <li><code>extern</code>で宣言した変数に対して、初期化式を記述しています。<code>extern</code>を無視します。</li> </ul> ⇒ <code>extern</code> を削除してください。             |
| far pointer (implicitly) casted by near pointer       | <ul style="list-style-type: none"> <li><code>far</code>ポインタが、<code>near</code>ポインタに変換されました。</li> </ul> ⇒ データの型、 <code>near/far</code> を確認してください。                      |
| function must be far                                  | <ul style="list-style-type: none"> <li>関数を<code>near</code>型で宣言しています。</li> </ul> ⇒ 正しく記述してください。                                                                       |
| function 関数名 has no-used argument (変数名)               | <ul style="list-style-type: none"> <li>関数引数に宣言された変数は使用されていません。</li> </ul> ⇒ 変数を確認してください。                                                                              |
| handler function called                               | <ul style="list-style-type: none"> <li><code>#pragma HANDLER</code>で指定した関数を呼び出しています。</li> </ul> ⇒ ハンドラ関数は呼び出さないようにしてください。                                             |
| handler function can't return value                   | <ul style="list-style-type: none"> <li><code>#pragma HANDLER</code>で指定した関数が、戻り値を使用しています。</li> </ul> ⇒ <code>#pragma HANDLER</code> で指定した関数は、戻り値を使用できません。戻り値を削除してください。 |
| handler function has argument                         | <ul style="list-style-type: none"> <li><code>#pragma HANDLER</code>で指定した関数が、引数を使用しています。</li> </ul> ⇒ <code>#pragma HANDLER</code> で指定した関数は、引数を使用できません。引数を削除してください。    |
| hex character is out of range                         | <ul style="list-style-type: none"> <li>文字定数においてHEX文字が長すぎます。また、<code>¥</code>の後に16進数以外の文字が入っています。</li> </ul> ⇒ HEX文字を短くしてください。                                         |
| identifier (メンバ名) is duplicated, this declare ignored | <ul style="list-style-type: none"> <li>メンバ名が重複して定義されています。この宣言を無視します。</li> </ul> ⇒ メンバ名の宣言を1つにしてください。                                                                  |
| identifier (変数名) is duplicated                        | <ul style="list-style-type: none"> <li>変数名が重複して定義されています。この宣言を無視します。</li> </ul> ⇒ 変数名の宣言を1つにしてください。                                                                    |
| identifier (変数名) is shadowed                          | <ul style="list-style-type: none"> <li>引数で宣言した変数名と同じ変数名の<code>auto</code>変数を使用しています。<code>auto</code>変数を無視します。</li> </ul> ⇒ 引数で使った変数名以外を使用してください。                     |
| illegal storage class for argument, 'extern' ignore   | <ul style="list-style-type: none"> <li>関数定義の引数リストにおいて、不当な記憶域クラスを使用しています。</li> </ul> ⇒ 正しい記憶域クラスを指定してください。                                                             |
| incomplete array access                               | <ul style="list-style-type: none"> <li>不完全型の多次元配列を参照しています。</li> </ul> ⇒ 多次元配列のサイズを明記してください。                                                                           |
| incompatible pointer types                            | <ul style="list-style-type: none"> <li>ポインタの示すオブジェクトの型が異なります。</li> </ul> ⇒ ポインタの型を確認してください。                                                                           |
| incomplete return type                                | <ul style="list-style-type: none"> <li>不完全な型を戻り値に記述しています。</li> </ul> ⇒ 戻り値を確認してください。                                                                                  |
| incomplete struct member                              | <ul style="list-style-type: none"> <li>不完全な構造体をメンバとして記述しています。</li> </ul> ⇒ 完全な構造体を記述してください。                                                                           |
| init elements overflow,ignored                        | <ul style="list-style-type: none"> <li>初期化式が初期化しようとする変数のサイズを超えました。</li> </ul> ⇒ 初期化式の数が、初期化する変数のサイズを超えないようにしてください。                                                    |

表F.23 ccom100ワーニングメッセージ一覧表 (5/9)

| ワーニングメッセージ                                                                     | ワーニング内容と対策                                                                                                                                                  |
|--------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| inline function is called as normal function before, change to static function | <ul style="list-style-type: none"> <li>記憶クラスinline で宣言された関数が通常の関数として呼び出されています。</li> </ul> ⇒ inline関数は使用する前に必ず定義を行ってください。                                    |
| integer constant is out of range                                               | <ul style="list-style-type: none"> <li>整数定数の値がunsigned longで表現できる値を超えました。</li> </ul> ⇒ 定数の値をunsigned longで表現できる値で記述してください。                                  |
| interrupt function called                                                      | <ul style="list-style-type: none"> <li>#pragma INTERRUPTで指定した関数を呼び出しています。</li> </ul> ⇒ 割り込み処理関数は呼び出さないようにしてください。                                            |
| interrupt function can't return value                                          | <ul style="list-style-type: none"> <li>#pragma INTERRUPTで指定した割り込み処理関数が、引数を使用しています。</li> </ul> ⇒ 割り込み関数では引数を使用できません。引数を削除してください。                             |
| interrupt function has argument                                                | <ul style="list-style-type: none"> <li>#pragma INTERRUPTで指定した割り込み処理関数が、引数を使用しています。</li> </ul> ⇒ 割り込み関数では引数を使用できません。引数を削除してください。                             |
| invalid #pragma EQU                                                            | <ul style="list-style-type: none"> <li>#pragma EQUの記述に誤りがあります。この行を無視します。</li> </ul> ⇒ 正しく記述してください。                                                          |
| invalid #pragma SECTION, unknown section base name                             | <ul style="list-style-type: none"> <li>#pragma SECTIONにおいてセクション名に誤りがあります。指定できるセクション名は data、bss、program、rom、interruptです。この行を無視します。</li> </ul> ⇒ 正しく記述してください。 |
| invalid #pragma operand, ignored                                               | <ul style="list-style-type: none"> <li>#pragmaのオペランドの記述に誤りがあります。この行を無視します。</li> </ul> ⇒ 正しく記述してください。                                                        |
| invalid function argument                                                      | <ul style="list-style-type: none"> <li>関数引数が正しく記述されていません。</li> </ul> ⇒ 関数引数を正しく記述してください。                                                                    |
| invalid return type                                                            | <ul style="list-style-type: none"> <li>return文の式が関数の型と異なっています。</li> </ul> ⇒ リターン値を関数の型に合わせるか、関数の型をリターン値の型に合わせてください。                                         |
| invalid storage class for function, change to extern                           | <ul style="list-style-type: none"> <li>関数宣言において、不当な記憶域クラスを使用しています。externとして処理します。</li> </ul> ⇒ 記憶域クラスをexternにしてください。                                        |
| Kanji in #pragma ADDRESS                                                       | <ul style="list-style-type: none"> <li>#pragma ADDRESSの記述に漢字コードが含まれています。この行を無視します。</li> </ul> ⇒ この宣言では漢字コードを記述しないでください。                                     |
| Kanji in #pragma BITADDRESS                                                    | <ul style="list-style-type: none"> <li>#pragma BITADDRESSの記述に漢字コードが含まれています。この行を無視します。</li> </ul> ⇒ この宣言では漢字コードを記述しないでください。                                  |
| keyword (キーワード) are reserved for future                                        | <ul style="list-style-type: none"> <li>将来のために予約されているキーワードを使用しています。</li> </ul> ⇒ 別の名前に変更してください。                                                              |
| large type was implicitly cast to small type                                   | <ul style="list-style-type: none"> <li>大きい型から小さい型への代入により、値の上位バイト(ワード)が失われる可能性があります。</li> </ul> ⇒ 型を確認してください。記述が正しい場合は、このワーニングは無視してください。                     |

表F.24 ccom100ワーニングメッセージ一覧表 (6/9)

| ワーニングメッセージ                                                                       | ワーニング内容と対策                                                                                                                                                                                                |
|----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mismatch prototyped parameter type                                               | <ul style="list-style-type: none"> <li>プロトタイプ宣言で宣言した時と引数の型が異なります。</li> </ul> ⇒ 引数の型を確認してください。                                                                                                             |
| meaningless statements deleted in optimize phase                                 | <ul style="list-style-type: none"> <li>無意味な記述が最適化で削除されました。</li> </ul> ⇒ 無意味な記述を削除してください。                                                                                                                  |
| meaningless statement                                                            | <ul style="list-style-type: none"> <li>文が "=" で終わっています。</li> </ul> ⇒ "=" と記述するところを誤って "==" と記述している可能性があります。故意にそう記述したものかを確認してください。                                                                         |
| mismatch function pointer assignment                                             | <ul style="list-style-type: none"> <li>レジスタ引数の関数のアドレスを、レジスタ引数でない(プロトタイプされていない)関数のポインタ変数へ代入しています。</li> </ul> ⇒ 関数のポインタ変数の宣言をプロトタイプ形式にしてください。                                                               |
| multi-character character constant                                               | <ul style="list-style-type: none"> <li>2文字以上の文字定数を使用しています。</li> </ul> ⇒ 2文字以上のときはワイド文字(L <code>xx</code> )を使用してください。                                                                                      |
| near/far is conflict beyond over typedef                                         | <ul style="list-style-type: none"> <li><code>near/far</code> を指定して <code>typedef</code> した型を参照時に再度、<code>near/far</code> を指定して宣言しています。</li> </ul> ⇒ 型指定子を正しく記述してください。                                     |
| No hex digit                                                                     | <ul style="list-style-type: none"> <li>16進数の定数に16進数で使用できない文字が含まれています。</li> </ul> ⇒ 16進数の定数は 0 から 9、A から F、a から f で記述してください。                                                                               |
| No initialized of 変数名                                                            | <ul style="list-style-type: none"> <li>レジスタ変数を初期化しないまま使用している可能性があります。</li> </ul> ⇒ レジスタ変数に対する代入を行なってください。                                                                                                 |
| No storage class & data type in declare, global storage class & int type assumed | <ul style="list-style-type: none"> <li>記憶域クラス指定子、型指定子なしに、変数を宣言しています。int として処理します。</li> </ul> ⇒ 記憶域クラス指定子、型指定子を記述してください。                                                                                   |
| non-initialized variable '変数名' is used                                           | <ul style="list-style-type: none"> <li>初期化していない変数を参照している可能性があります。</li> </ul> ⇒ 記述を確認してください。このワーニングは、関数の最後の行で発生することもあります。この場合、関数内の <code>auto</code> 変数等の記述を確認してください。記述が正しい場合は、このワーニングは無視してください。           |
| non-prototyped function used                                                     | <ul style="list-style-type: none"> <li>プロトタイプ宣言していない関数を呼び出しています。コンパイルオプション-W<code>non_prototype</code>を指定した時のみ出力されます。</li> </ul> ⇒ 関数のプロトタイプ宣言を記述するか、コンパイルオプション-W <code>non_prototype</code> を指定しないでください。 |
| non-prototyped function declared                                                 | <ul style="list-style-type: none"> <li>定義されている関数のプロトタイプ宣言が存在しません(コンパイルオプション-W<code>NP</code>指定時のみ表示)。</li> </ul> ⇒ プロトタイプ宣言を行ってください。                                                                      |
| octal constant is out of range                                                   | <ul style="list-style-type: none"> <li>8進数の定数に8進数で使用できない文字が含まれています。</li> </ul> ⇒ 8進数の定数は0から7で記述してください。                                                                                                    |
| octal character is out of range                                                  | <ul style="list-style-type: none"> <li>8進数の定数に8進数で使用できない文字が含まれています。</li> </ul> ⇒ 8進数の定数は0から7で記述してください。                                                                                                    |
| overflow in floating value converting to integer                                 | <ul style="list-style-type: none"> <li>整数型には格納できない巨大な浮動小数点値を、整数型に代入しています。</li> </ul> ⇒ 代入式を再確認してください。                                                                                                     |

表F.25 ccom100ワーニングメッセージ一覧表 (7/9)

| ワーニングメッセージ                                                           | ワーニング内容と対策                                                                                                                              |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| old style function declaration                                       | <ul style="list-style-type: none"> <li>ANSI(ISO)C 以前の形式で関数定義を記述しています。</li> </ul> ⇒ ANSI(ISO)形式で 関数定義を記述してください。                          |
| prototype function is defined as non-prototyped function before      | <ul style="list-style-type: none"> <li>プロトタイプ宣言していない関数を再度プロトタイプ宣言しています。</li> </ul> ⇒ 関数の宣言方法を統一してください。                                  |
| redefined type                                                       | <ul style="list-style-type: none"> <li>typedefで、定義済みの型名を再定義しています。</li> </ul> ⇒ 別の型名を使用するか、記述ミスがないか確認してください。                             |
| redefined type name of (識別子)                                         | <ul style="list-style-type: none"> <li>typedefで同じ識別子名を2回以上定義しています。</li> </ul> ⇒ 識別子名を正しく記述してください。                                       |
| redefined type name of (識別子)                                         | <ul style="list-style-type: none"> <li>typedefで同じ識別子名を2回以上定義しています。</li> </ul> ⇒ 識別子名を正しく記述してください。                                       |
| register parameter function used before as stack parameter function  | <ul style="list-style-type: none"> <li>レジスタ引数の関数が以前にスタック引数の関数として使用しています。</li> </ul> ⇒ 関数を使用する前にプロトタイプ宣言を行ってください。                        |
| RESTRICT qualifier can set only pointer type.                        | <ul style="list-style-type: none"> <li>RESTRICT修飾子がポインタ以外で宣言されています。</li> </ul> ⇒ ポインタのみに宣言してください。                                       |
| section name 'interrupt' no more used                                | <ul style="list-style-type: none"> <li>#pragma SECTIONで指定されたセクション名に'interrupt'を使用しています。</li> </ul> ⇒ 'interrupt'は使用できません。別の名称に変更してください。 |
| size of incomplete type                                              | <ul style="list-style-type: none"> <li>sizeof演算子のオペランドに定義されていない構造体、共用体を記述しています。</li> </ul> ⇒ 構造体、共用体を先に定義してください。                        |
| size of incomplete array type                                        | <ul style="list-style-type: none"> <li>sizeof演算子のオペランドに定義されている配列の要素数が決定していません。</li> </ul> ⇒ 構造体、共用体を先に定義してください。                         |
| size of void                                                         | <ul style="list-style-type: none"> <li>voidのサイズを求めています。無効なサイズです。</li> </ul> ⇒ voidのサイズは求められません。                                         |
| standard library "関数名()" need "インクルードファイル名"                          | <ul style="list-style-type: none"> <li>標準ライブラリ関数をヘッダファイルをインクルードしないで使っています。</li> </ul> ⇒ ヘッダファイルをインクルードしてください。                           |
| static variable in inline function                                   | <ul style="list-style-type: none"> <li>記憶クラスinline で宣言した関数内でstaticデータを宣言しています。</li> </ul> ⇒ インライン関数内で、staticデータを宣言しないでください。             |
| string size bigger than array size                                   | <ul style="list-style-type: none"> <li>初期化する変数のサイズより、初期化式のサイズが大きい。</li> </ul> ⇒ 初期化式のサイズは、変数と同じか、変数より小さくしてください。                         |
| string terminator not added                                          | <ul style="list-style-type: none"> <li>配列の要素数と初期化式のサイズが同じであるため、文字列の最後に付加する'\0'は、付加しません。</li> </ul> ⇒ 配列の要素数を増やしてください。                   |
| struct (or union) member's address can't has no near far information | <ul style="list-style-type: none"> <li>構造体(もしくは共用体)のメンバー(変数)の配置位置情報としてnear, far を指定しています。</li> </ul> ⇒ メンバーは near, far を指定しないでください。     |



表F.26 ccom100ワーニングメッセージ一覧表 (8/9)

| ワーニングメッセージ                                                  | ワーニング内容と対策                                                                                                                             |
|-------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| task function called                                        | <ul style="list-style-type: none"> <li>• #pragma TASKで指定した関数を呼び出しています。</li> </ul> ⇒ タスク関数は呼び出さないようにしてください。                             |
| task function can't return value                            | <ul style="list-style-type: none"> <li>• #pragma TASKで指定した関数が、戻り値を使用しています。</li> </ul> ⇒ #pragma TASKで指定した関数は、戻り値を使用できません。戻り値を削除してください。 |
| task function has invalid argument                          | <ul style="list-style-type: none"> <li>• #pragma TASK で指定した関数が、引数を使用しています。</li> </ul> ⇒ #pragma TASK で指定した関数は、引数を使用できません。引数を削除してください。  |
| this comparison is always false                             | <ul style="list-style-type: none"> <li>• 常に偽になる比較を行っています。</li> </ul> ⇒ 条件式を確認してください。                                                   |
| this comparison is always true                              | <ul style="list-style-type: none"> <li>• 常に真になる比較を行っています。</li> </ul> ⇒ 条件式を確認してください。                                                   |
| this feature not supported now, ignored                     | <ul style="list-style-type: none"> <li>• 文法エラーです。この記述は、将来拡張用の文法ですので使用しないでください。</li> </ul> ⇒ 正しく記述してください。                               |
| this function used before with non-default argument         | <ul style="list-style-type: none"> <li>• 使用されたことのある関数をデフォルト引数を持つ関数として宣言しています。</li> </ul> ⇒ 関数を使用する前にデフォルト引数を宣言してください。                  |
| this interrupt function is called as normal function before | <ul style="list-style-type: none"> <li>• 使用したことのある関数を #pragma INTERRUPTで宣言していません。</li> </ul> ⇒ 割り込み関数は呼び出せません。#pragma の内容を確認してください。    |
| too big octal character                                     | <ul style="list-style-type: none"> <li>• 文字定数、文字列中の8進数の定数が、限界値(10進数で255)を超えました。</li> </ul> ⇒ 255以下の値で記述してください。                         |
| too few parameters                                          | <ul style="list-style-type: none"> <li>• プロトタイプ宣言で宣言した時より引数が足りません。</li> </ul> ⇒ 引数の数を確認してください。                                         |
| too many parameters                                         | <ul style="list-style-type: none"> <li>• プロトタイプ宣言で宣言した時より引数が多すぎます。</li> </ul> ⇒ 引数の数を確認してください。                                         |
| unknown #pragma STRUCT xxx                                  | <ul style="list-style-type: none"> <li>• #pragma STRUCTxxxは、処理できません。この行を無視します。</li> </ul> ⇒ 正しく記述してください。                               |
| Unknown debug option (-dx)                                  | <ul style="list-style-type: none"> <li>• コンパイルオプション-dxは、指定できません。</li> </ul> ⇒ コンパイルオプションを正しく指定してください。                                  |
| Unknown function option (-Wxxx)                             | <ul style="list-style-type: none"> <li>• コンパイルオプション-Wxxxは、指定できません。</li> </ul> ⇒ コンパイルオプションを正しく指定してください。                                |
| Unknown function option (-fx)                               | <ul style="list-style-type: none"> <li>• コンパイルオプション-fxは、指定できません。</li> </ul> ⇒ コンパイルオプションを正しく指定してください。                                  |
| Unknown function option (-gx)                               | <ul style="list-style-type: none"> <li>• コンパイルオプション-gxは、指定できません。</li> </ul> ⇒ コンパイルオプションを正しく指定してください。                                  |
| Unknown optimize option (-mx)                               | <ul style="list-style-type: none"> <li>• コンパイルオプション-mxは、指定できません。</li> </ul> ⇒ コンパイルオプションを正しく指定してください。                                  |
| Unknown optimize option (-Ox)                               | <ul style="list-style-type: none"> <li>• コンパイルオプション-Oxは、指定できません。</li> </ul> ⇒ コンパイルオプションを正しく指定してください。                                  |

表F.27 ccom100ワーニングメッセージ一覧表 (9/9)

| ワーニングメッセージ                               | ワーニング内容と対策                                                                                                                                                          |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unknown option (-x)                      | <ul style="list-style-type: none"> <li>コンパイルオプション-xは、指定できません。</li> </ul> ⇒ コンパイルオプションを正しく指定してください。                                                                  |
| unknown pragma pragma-指示 used            | <ul style="list-style-type: none"> <li>サポートされていない #pragma を記述しています。</li> </ul> ⇒ #pragma の内容を確認してください。この警告は コンパイルオプション-Wunknown_pragma (-WUP)、および -Wall指定時のみ表示されます。 |
| wchar_t array initialized by char string | <ul style="list-style-type: none"> <li>wchar_t型の初期化式をchar型の文字列で初期化しています。</li> </ul> ⇒ 初期化式の型を合わせてください。                                                              |
| zero divide in constant folding          | <ul style="list-style-type: none"> <li>除算演算子、剰余算演算子において除数が0です。</li> </ul> ⇒ 除数は、0以外を使用してください。                                                                       |
| zero divide, ignored                     | <ul style="list-style-type: none"> <li>除算演算子、剰余算演算子において除数が0です。</li> </ul> ⇒ 除数は、0以外を使用してください。                                                                       |
| zero width for bitfield                  | <ul style="list-style-type: none"> <li>ビットフィールドの幅が0です。</li> </ul> ⇒ ビット幅を1以上で記述してください。                                                                              |
| no const in previous declaration         | <ul style="list-style-type: none"> <li>const 修飾子がない関数、変数の宣言に対して、実体定義された関数、変数で、const 修飾しています。</li> </ul> ⇒ 関数、変数の宣言とその実体定義で、const 修飾を統一してください。                       |
| xxx was declared but never referenced    | <ul style="list-style-type: none"> <li>参照されない宣言があります。</li> </ul> ⇒ 宣言を削除してください。                                                                                     |



---

## 付録G Call Walker 用スタック情報ファイル作成ツール gensni 操作方法

---

High-performance Embedded Workshop のスタック解析ツール Call Walker を使用するためには、入力ファイルとしてスタック情報ファイル（拡張子.sni のファイル）が必要です。

スタック情報ファイルは、アブソリュートモジュールファイルから Call Walker 用スタック情報ファイル作成ツール gensni を使用して生成します。

### G.1 Call Walkerの起動

Call Walker は、High-performance Embedded Workshop に登録されている"Call Walker"を選択、または、High-performance Embedded Workshop の Tools メニューからの選択によって起動します。

Call Walker 起動後は、[File]メニューの[Import Stack File...]から入力ファイルとしてスタック情報ファイルを指定してください。

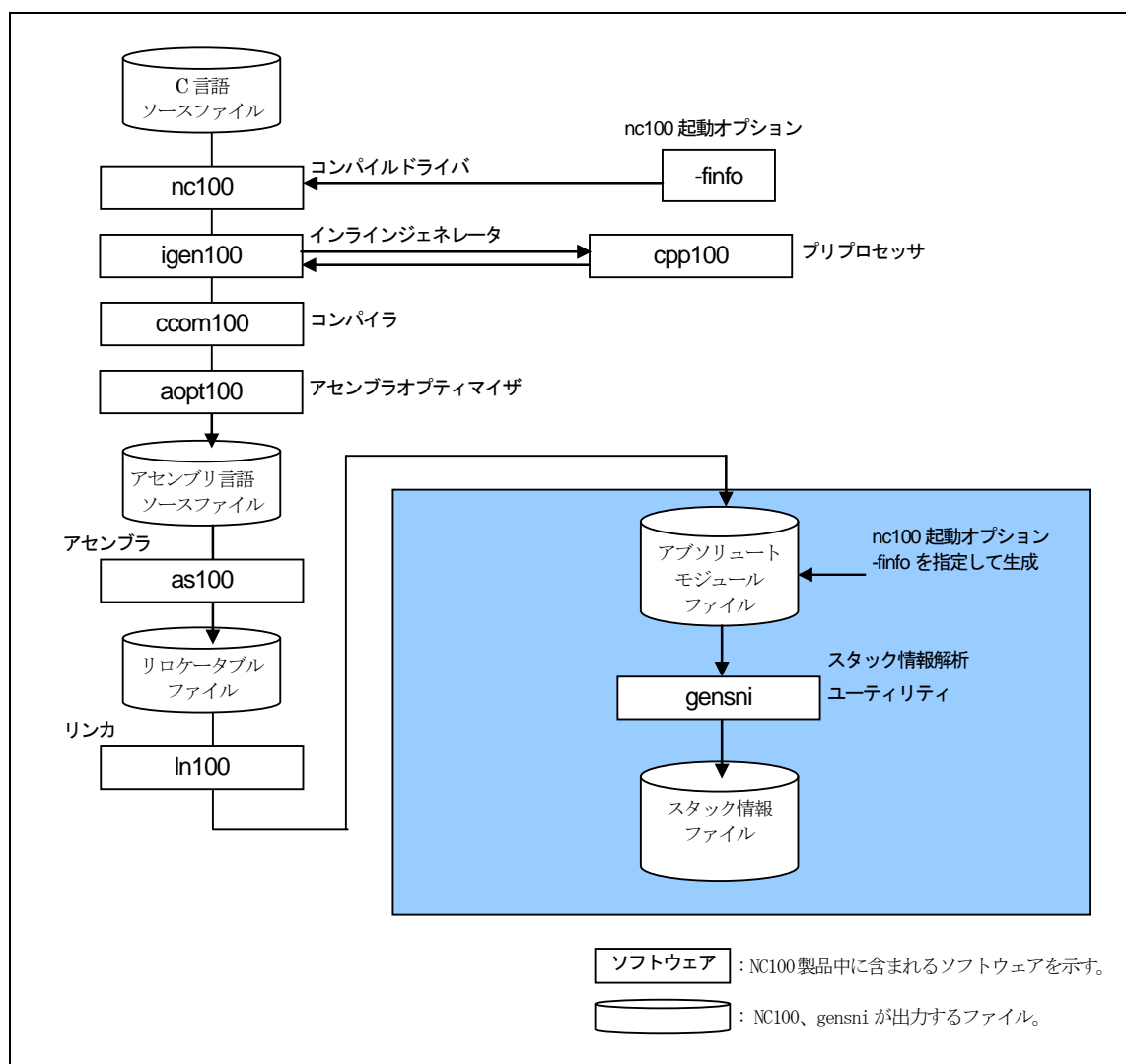
### G.2 gensniの概要

#### G.2.1 gensniの処理概要

gensni は、Call Walker 用.sni ファイルを作成するためのツールです。

gensni は、アブソリュートモジュールファイル（拡張子.x30）を処理して、スタック情報ファイル（拡張子.sni）ファイルを生成します。gensni を使用するには、コンパイル時にコンパイルオプション “-finfo -g” を指定してアブソリュートモジュールファイル（拡張子.x30）を生成してください。

NC100 の処理フローを【図G.1】示します。



図G.1 NC100の処理フロー

### G.3 gensniの起動方法

High-performance Embedded Workshop から Call Walker を起動した場合、gensni は自動的に実行されます。

しかし、High-performance Embedded Workshop 以外から Call Walker を起動する場合は、gensni は自動的に実行されません。この場合は、Windows のコマンドプロンプト上で gensni を起動してください。

#### G.3.1 入力書式

gensni を起動するためには、以下の図に示す入力書式にしたがって、入力ファイル名、起動オプションを指定します。

```

% gensni△[起動オプション] △アブソリュートモジュールファイル (拡張子.x30)

% : プロンプトを示します。
< > : 必須項目を示します。
[] : 必要に応じて記述する項目を示します。
△ : スペースを示します。
複数の起動オプションを記述する場合はスペースで区切ってください。

```

図G.2 gensni コマンドの入力書式

gensni を使用するためには、本コンパイラの起動オプションに、

- インспекタ情報の出力.....-finfoオプション
- デバッグ情報の出力.....-gオプション

の両方を指定して、アブソリュートモジュールファイル(拡張子 .x30)を生成してください。

以下に入力例を示します。入力例では gensni に、以下のオプションを指定しています。

- 出力情報の指定ファイルへ出力.....oオプション

(デフォルトでは、入力ファイルの.x30 を.sni に変更したファイルに出力します。)

#### アブソリュートモジュールファイルの生成:

```

%nc100 -g -finfo ncrt0.a30 sample.c<RET>
R32C/100 Series C Compiler V.X.XX Release XX
Copyright(C) XXXX(XXXX-XXXX). Renesas Electronics Corp.
and Renesas Solutions Corp., All rights reserved.
ncrt0.a30
sample.c

```

%

#### スタック情報ファイルの生成:

```

%gensni -o sample ncrt0.x30<RET>

```

```

sample.sni is created.

```

%

```

<RET> : リターンキーの入力を示します

```

図G.3 gensni コマンドの入力例

## G.3.2 オプションリファレンス

gensniの起動オプションを【表G.1】に示します。

表G.1 gensniの起動オプション

| オプション    | 短縮形 | 機能                                                                                                                                                                                                                                 |
|----------|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -o ファイル名 | なし  | スタック情報ファイルの名前を指定します。<br><ul style="list-style-type: none"> <li>● 本オプションが指定されていない場合、スタック情報ファイル名は入力ファイル名の拡張子を.sniに指定した名前になります。</li> <li>● 拡張子を含めてファイル名の指定をした場合、生成するファイルは拡張子を含め指定されたファイル名で生成します。拡張子の指定がない場合、拡張子は.sniになります。</li> </ul> |
| -V       | なし  | gensniの起動メッセージを表示して処理を終了します。<br>スタック情報ファイルは作成しません。                                                                                                                                                                                 |

### -o スタック情報ファイル名

スタック情報ファイル名を指定する

- 機能
- 本オプションが指定されていない場合、スタック情報ファイル名は入力ファイル名の拡張子を.sniに変更した名前となります。
  - スタック情報ファイル名に拡張子を指定した場合は.sniを指定した拡張子に変更します。スタック情報ファイル名に拡張子の指定がない場合は、拡張子を.sniとします。

補足説明: 本オプションを使用することにより、スタック情報ファイル名を任意に変更できます。拡張子の変更も可能です。

### -V

gensniの起動メッセージを表示して処理を終了する

- 機能
- gensniの起動メッセージを表示して処理を終了します。
  - スタック情報ファイルは作成しません。

## G.4 gensniのエラーメッセージ

### G.4.1 エラーメッセージ

【表G.2】にgensni が出力するエラーメッセージとその内容及び対処方法を示します。

表G.2 gensni エラーメッセージ一覧表

| エラーメッセージ                                | エラー内容と対策                                                                                                                                                                                      |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| usage: gensni [-V][-o out_file] in_file | <ul style="list-style-type: none"> <li>入力書式に間違いがあります。</li> </ul> ⇒ 正しい入力書式を指定してください。                                                                                                          |
| Can't open file: XXX                    | <ul style="list-style-type: none"> <li>アブソリュートモジュールファイルがオープンできません。</li> </ul> ⇒ ファイルの有無、ファイルの属性を確認してください。                                                                                     |
| Can't create file: XXX                  | <ul style="list-style-type: none"> <li>スタック情報ファイルを作成できません。</li> </ul> ⇒ ファイル、フォルダの属性を確認してください。<br>デスクの空き容量を確認してください。                                                                          |
| Illegal file format: XXX                | <ul style="list-style-type: none"> <li>アブソリュートモジュールファイルの内容に間違いがあります。スタック情報ファイルを作成できません。</li> </ul> ⇒ アブソリュートモジュールファイルがNC100で作成したファイルであるか確認してください。<br>また、コンパイラオプションに-finfo、-gを指定しているか確認してください。 |
| Not enough memory                       | <ul style="list-style-type: none"> <li>メモリが確保できませんでした。</li> </ul> ⇒ PCのメモリ容量を確認してください。                                                                                                        |

---

R32C/100 シリーズ用 C コンパイラパッケージ V.1.02  
C コンパイラユーザーズマニュアル

発行年月日: 2010 年 4 月 1 日 Rev.2.00

発行: 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

編集: 株式会社ルネサス ソリューションズ

---

© 2010 Renesas Electronics Corporation, All rights reserved. Printed in Japan.

R32C/100 シリーズ用  
C コンパイラパッケージ V.1.02  
C コンパイラ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

RJJ10J2504-0200