お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010 年 4 月 1 日
ルネサスエレクトロニクス株式会社

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

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

■本ドキュメントに記載されているURLは、以下のとおり読み替えをお願いいたします。
http://www.necel.com/
http://www2.renesas.com/
↓
開発環境トップページ　http://japan.renesas.com/tools
ダウンロードポータル　http://japan.renesas.com/tool_download

■技術問合せについては、以下のページをご覧ください。
http://japan.renesas.com/tech_inquiry

■ツールユーザ登録については、以下のページをご覧ください。
http://japan.renesas.com/myrenesas

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が、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、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ージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、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がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

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

RENESAS

ユーザーズ・マニュアル

CC78K0R Ver.1.00

Cコンパイラ

操作編

対象デバイス
78K0Rマイクロコントローラ

資料番号　U17838JJ1V0UM00（第1版）
発行年月　June 2006　CP(K)

〔メ　モ〕

目次要約

Windows**および**WindowsXP**は，米国**Microsoft Corporation**の米国およびその他の国における登録商標または商標です。**

PC/AT**は，米国**IBM Corp.**の商標です。**

i386**は，米国**Intel Corporation**の商標です。**

- **本資料に記載されている内容は**2006**年**6**月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。**
- **文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。**
- **当社は，本資料に記載された当社製品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，一切その責を負いません。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。**
- **本資料に記載された回路，ソフトウエアおよびこれらに関する情報は，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，ソフトウエアおよびこれらに関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，一切その責を負いません。**
- **当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等安全設計を行ってください。**
- **当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「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。**

  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**

  特別水準：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

  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，生命維持のための装置またはシステム等

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。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

(注)

（1）本事項において使用されている「当社」とは，NEC**エレクトロニクス株式会社および**NEC**エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいう。**

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E 02.11

はじめに

このマニュアルは，CC78K0R（78K0Rマイクロコントローラ　Cコンパイラ）についての機能および操作方法を正しく理解していただくことを目的としています。

このマニュアルでは，CC78K0Rのソース・プログラムの記述方法に関する説明はいたしません。したがって，このマニュアルをお読みになる前に，“CC78K0R　Cコンパイラ　ユーザーズ・マニュアル　言語編（U17837J）”（以降“言語編”とします）をお読みください。

【ターゲット・デバイス】

CC78K0Rでは，78K0Rマイクロコントローラ・マイクロコンピュータのソフトウエア開発が可能です。ご使用の際には，RA78K0R（78K0Rマイクロコントローラ　アセンブラ・パッケージ）（別売），ターゲットの種類に応じたデバイス・ファイルが必要となります。

【対 象 者】

このマニュアルは，デバイスのユーザーズ・マニュアル一読程度の知識があり，ソフトウエア・プログラミングの経験がある方を対象として書かれていますが，Cコンパイラ やC言語の知識は特に必要ありませんので，Cコンパイラをはじめて使われる方でもお読みいただけます。

【構　　成】

このマニュアルの構成を次に示します。

第1章　概　　説

マイクロコンピュータの開発における本Cコンパイラの役割，位置付けなどについて説明します。

第2章　製品概要とインストール方法

本Cコンパイラのインストール方法，提供するプログラムのファイル名，プログラムの動作環境などについて説明します。

第3章　コンパイルからリンクまでの手順

サンプル・プログラムを使用して，本Cコンパイラを実行する手順，コンパイル～リンク例などについて説明します。

第4章　CC78K0Rの機能

本Cコンパイラの最適化方法とROM化機能について説明します。

第5章　コンパイラ・オプション

コンパイラ・オプションの機能と指定方法，優先順位などについて説明します。

第6章　Cコンパイラの出力ファイル

本Cコンパイラが出力する各種リスト・ファイルの出力項目について説明します。

第7章　Cコンパイラの活用法

本Cコンパイラを上手に使うための方法を紹介します。

第8章　スタートアップ・ルーチン

本Cコンパイラは，スタートアップ・ルーチンをサンプルとして提供しています。スタートアップ・ルーチンおよびスタートアップ・ルーチン改良のポイントなどについて説明します。

第9章　エラー・メッセージ

本Cコンパイラが出力するエラー・メッセージについて説明します。

付　録

付録として，サンプル・プログラム，使用上の注意事項，コマンド・オプションがあります。

【読 み 方】

まず，実際に本Cコンパイラを使ってみたい方は，**第3章　コンパイルからリンクまでの手順**をお読みください。

Cコンパイラの一般的な知識のある方や言語編を読まれた方であれば，**第1章　概　説**を読み飛ばされても結構です。

【関連資料】

このマニュアルに関連する資料（ユーザーズ・マニュアルなど）を紹介します。関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

開発ツールの資料（ユーザーズ・マニュアル）

資　料　名		資料番号	
		和　　文	英　　文
CC78K0R Ver.1.00　Cコンパイラ	操作編	このマニュアル	U17838E
	言語編	U17837J	U17837E
RA78K0R Ver.1.00　アセンブラ・パッケージ	操作編	U17836J	U17836E
	言語編	U17835J	U17835E
SM+　システム・シミュレータ	操作編	U18010J	U18010E （作成予定）
PM+ Ver.6.20		U17990J	U17990E
ID78K0R-QB Ver.3.20　統合ディバッガ	操作編	U17839J	U17839E

【凡　例】

このマニュアル中で共通に使用される記号などの意味を示します。

RTOS：78K0**マイクロコントローラ用 リアルタイム**OS　RX78K0

…　：同一の形式を繰り返す

[　]　：[　] 内は省略可能

「　」　：「　」で囲まれた文字そのもの

“　”　：“　”で囲まれた文字そのもの

‘　’　：‘　’で囲まれた文字そのもの

太文字：文字そのもの

＿　：重要箇所，使用例での下線は入力文字列

△　：1文字以上の空白

⋮　：プログラム記述の省略形

(　)　：(　) で囲まれた文字そのもの

／　：区切り記号

＼　：バック・スラッシュ

UNIX™**の場合，‘￥’は‘＼（バック・スラッシュ）’となります。**

【ファイル名の規則】

コマンド行で指定する入力ファイルの指定規則を次に示します。

(1) ディスク型ファイル名指定

[ドライブ名]　[¥]　[[パス名]…] プライマリ・ネーム [. [ファイル・タイプ]]

①　②　③　④　⑤

①　ファイルを格納しているドライブ名（A: ～ Z:）を指定します。

②　ルート・ディレクトリ名を指定します。

③　サブディレクトリ名を指定します。

OSが許す長さの文字列を指定します。

使用可能な文字：

OSが許している文字から，括弧（ () ），セミコロン（ ; ），コンマ（ , ）を除いた文字とします。

ただし，ハイフン（-）をパス名の先頭に使用することはできません。

④　プライマリ・ネーム

OSが許す長さの文字列を指定します。

使用可能な文字：

OSが許している文字から，括弧（ () ），セミコロン（ ; ），コンマ（ , ）を除いた文字とします。

ただし，ハイフン（-）をファイル名の先頭に使用することはできません。

⑤　ファイル・タイプ

OSが許す長さの文字列を指定します。

使用可能な文字：

OSが許している文字から，括弧（ () ），セミコロン（ ; ），コンマ（ , ）を除いた文字とします。

例 C:¥nectools32¥smp78K0¥cc78K0¥prime.c

備考1. **‘：’，‘．’，‘￥’の前後に空白は指定できません。**

2. **英大文字と小文字の区別はされません。**

3. UNIX**の場合，‘￥’は‘＼（バック・スラッシュ）’となります。**

(2) デバイス型ファイル名指定

論理デバイスとして次のものがあります。

論理デバイス	説　明
CON	コンソールへ出力します。
PRN	プリンタへ出力します。
AUX	補助出力装置へ出力します。
NUL	ダミー出力（何も出力しません）。

目次

図の目次

図番号　タイトル, ページ

表の目次

表番号　タイトル, ページ

第 1 章 概説

78K0R シリーズ用 C コンパイラ（CC78K0R）は，ANSI-C [注]，または 78K0R シリーズの C 言語で記述された C ソース・プログラムを 78K0R シリーズ用の機械語に変換するプログラムです。

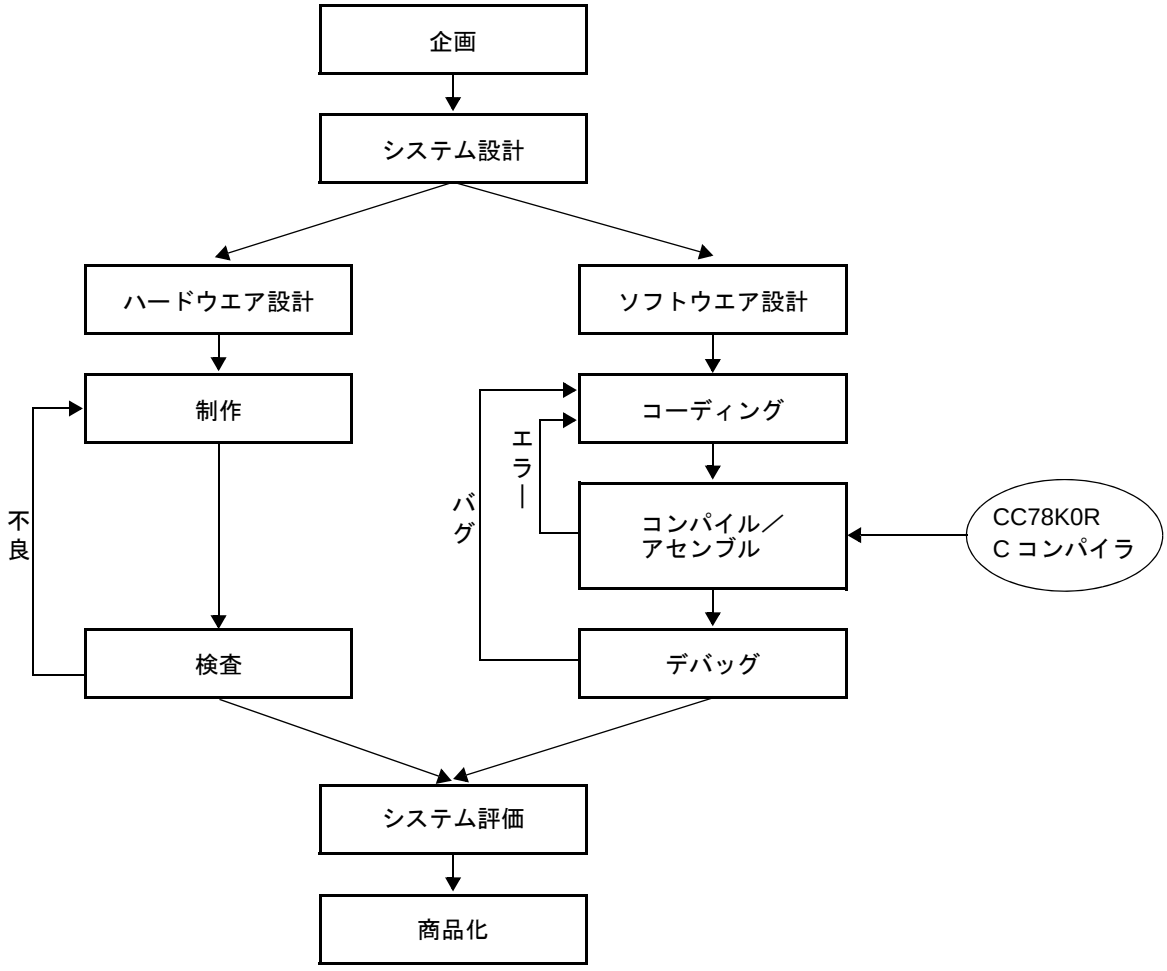
CC78K0R は，78K0R シリーズ用アセンブラ・パッケージ（RA78K0R）に添付されているプロジェクト・マネージャ（PM+）を利用することにより，Windows® 上で起動できるようになっています。PM+ を利用しない場合には，コマンド・プロンプト上で起動することが可能です。

注 ANSI-C とは，American National Standards Institute の規格に準拠した C 言語です。

1.1 CC78K0R の役割

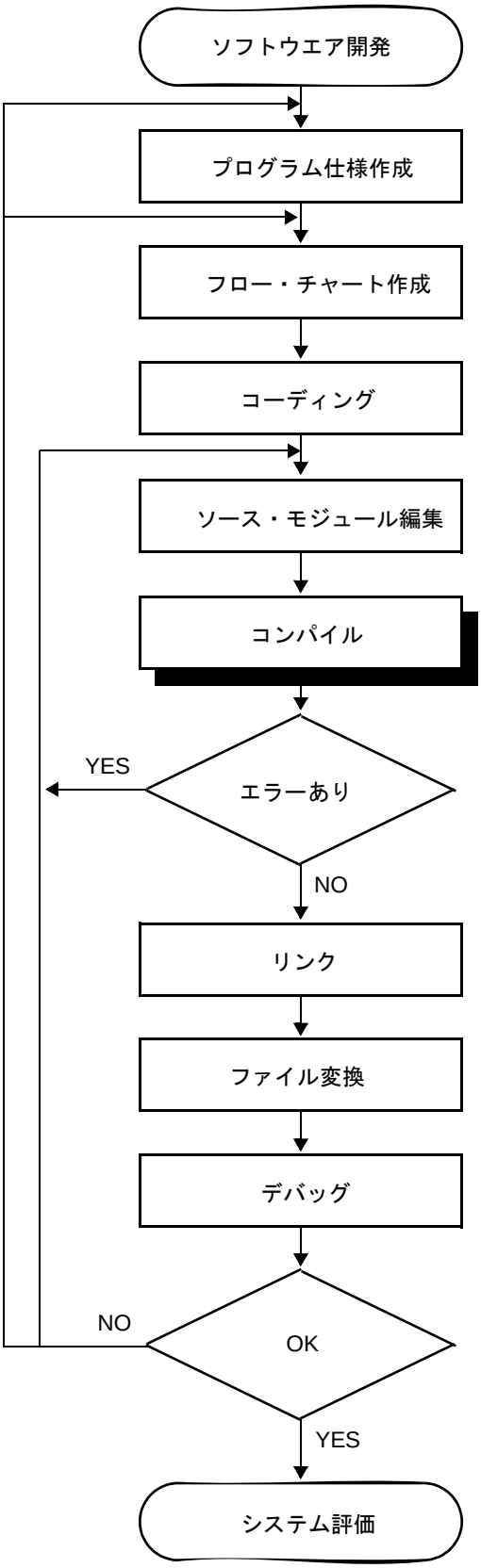
製品開発における，CC78K0R の位置付けを次に示します。

図 1-1 開発工程



ソフトウエアの開発工程を次に示します。

図 1-2 ソフトウエア開発工程



・・・ 78K0R シリーズ C 言語，または ANSI-C によります。

・・・ エディタを使い C ソース・モジュール・ファイルを作成します。

・・・ 参照ライブラリ，関数ライブラリと統合します。

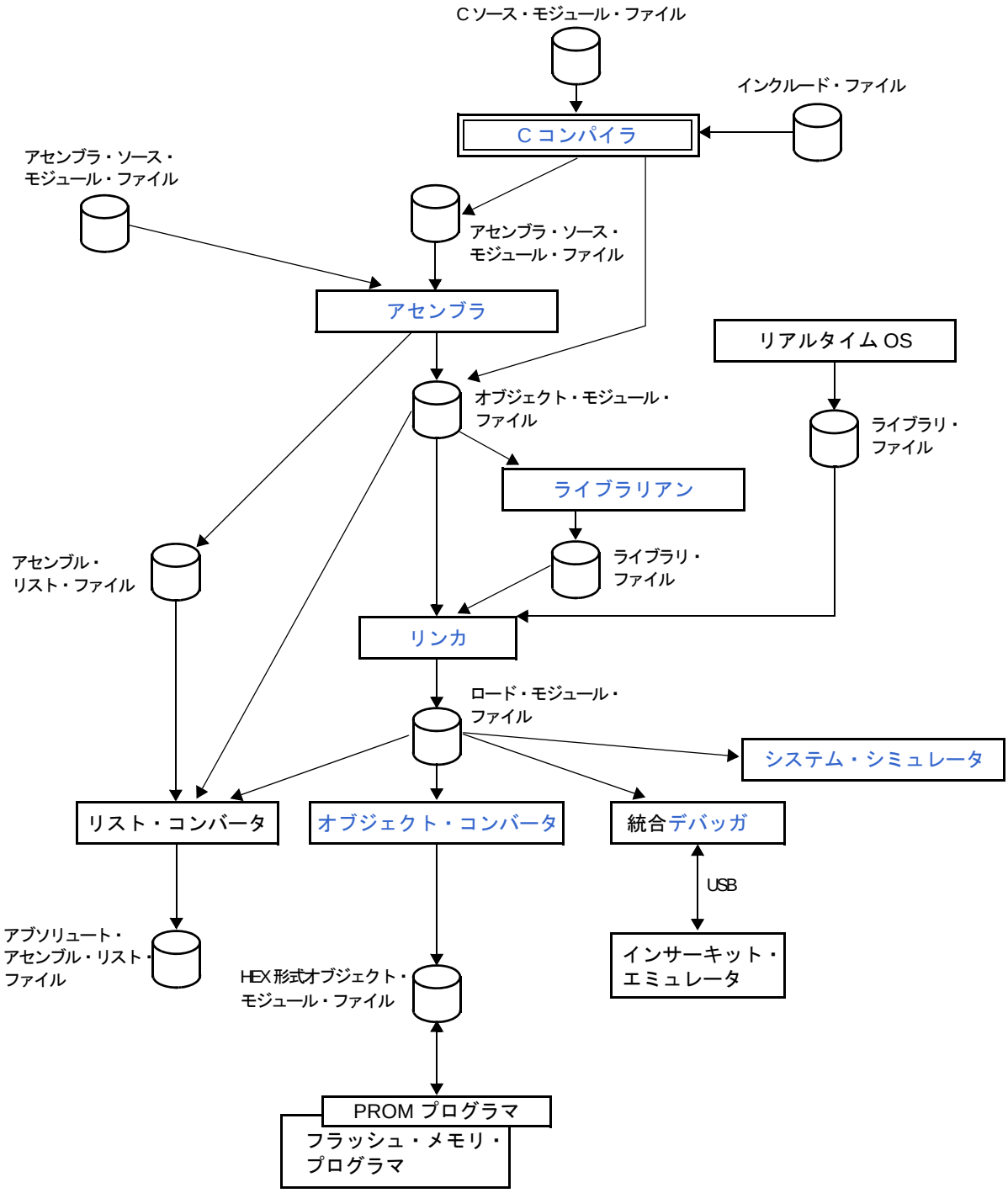
・・・ ファイルを HEX 形式に変換します。

・・・ ハードウエア・デバッガ（インサーキット・エミュレータなど）を使って動作確認します。

1.2 CC78K0R による開発手順

CC78K0R における開発手順を次に示します。

図 1-3 CC78K0R によるプログラム開発手順



1.2.1 エディタによるソース・モジュール・ファイルの作成

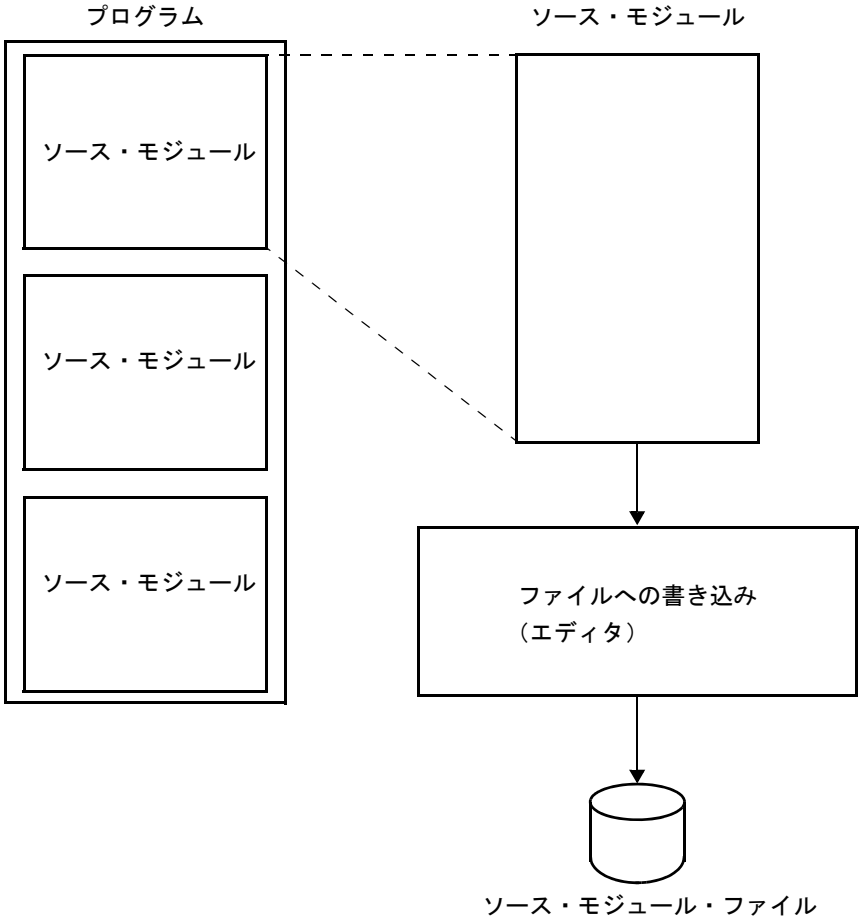
1 つのプログラムを機能的にいくつかのモジュールに分割します。

1 つのモジュールは，コーディングの単位になるもので，また，C コンパイラの入力単位にもなります。C コンパイラの入力単位となるモジュールを，C ソース・モジュールと呼びます。

各 C ソース・モジュールのコーディング終了後，エディタを使用してソース・モジュールをファイルに保存します。こうしてできたファイルを C ソース・モジュール・ファイルと呼びます。

C ソース・モジュール・ファイルは，CC78K0R への入力ファイルとなります。

図 1-4 ソース・モジュール・ファイルの作成



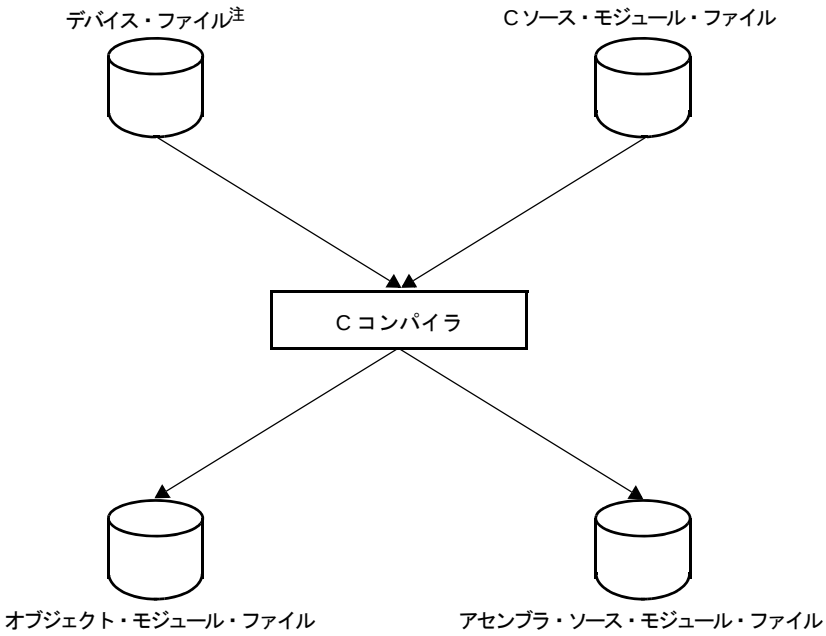
1.2.2 C コンパイラ

C コンパイラは，C ソース・モジュール・ファイルを入力し，C 言語から機械語へと変換します。

C ソース・モジュール・ファイル中に記述ミスを発見した場合は，コンパイル・エラーを出力します。コンパイル・エラーが発生しなかった場合には，オブジェクト・モジュール・ファイルを出力します。

また，アセンブリ言語レベルでのプログラム修正，および確認を行えるようにアセンブラ・ソース・モジュール・ファイルを出力することもできます。アセンブラ・ソース・モジュール・ファイルを出力したい場合には，コンパイルの際に -a オプション，または -sa オプションを指定してください（オプションについては，「第 5 章 コンパイラ・オプション」を参照してください）。

図 1-5 C コンパイラの機能



注 デバイス・ファイルはオンライン・デリバリ・サービス（ODS）から別途入手してください。
下記 URL から取得可能です。

http://www.necel.com/micro/ods/jpn/index.html

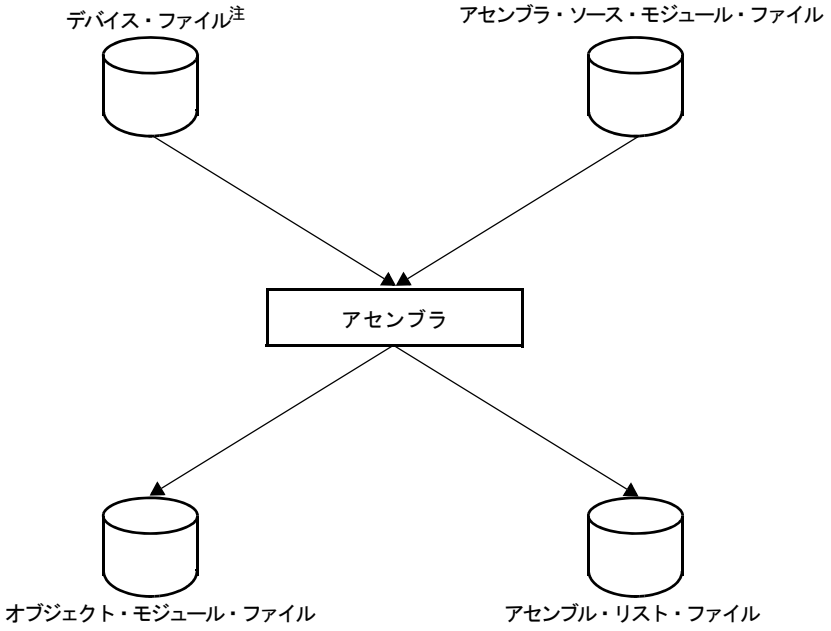
1.2.3 アセンブラ

アセンブルは，RA78K0R　アセンブラ・パッケージ（別売）に含まれているアセンブラを用いて行います。

アセンブラは，アセンブラ・ソース・モジュール・ファイルを入力し，アセンブリ言語から機械語へと変換します。

アセンブラ・ソース・モジュール・ファイル中に記述ミスを発見した場合は，アセンブル・エラーを出力します。アセンブル・エラーが発生しなかった場合には，機械語情報と各機械語がメモリ中のどのアドレスに配置されるべきかなどの配置情報を含むオブジェクト・モジュール・ファイルを出力します。また，アセンブラは，アセンブル時の情報をアセンブル・リスト・ファイルとして出力します。

図 1-6 アセンブラの機能



注 デバイス・ファイルはオンライン・デリバリ・サービス（ODS）から別途入手してください。
下記 URL から取得可能です。

http://www.necel.com/micro/ods/jpn/index.html

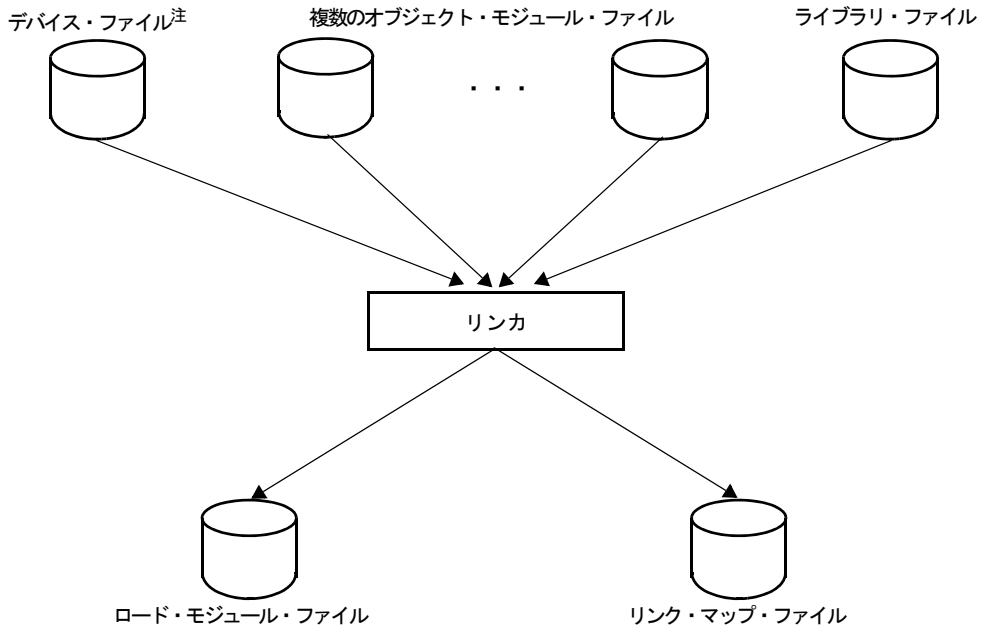
1.2.4 リンカ

リンクは，RA78K0R　アセンブラ・パッケージ（別売）に含まれているリンカを用いて行います。

リンカは，C コンパイラの出力したオブジェクト・モジュール・ファイル，またはアセンブラの出力したオブジェクト・モジュール・ファイルを複数個入力し，それらをライブラリ・ファイルと結合します（オブジェクト・モジュールが 1 つの場合でも，リンクしなければなりません）。そして，1 つのロード・モジュール・ファイルを出力します。

この際，リンカは，入力されたモジュール中のリロケータブルなセグメントの配置アドレスを決定します。これにより，リロケータブル・シンボルや外部参照シンボルの値を決定し，ロード・モジュール・ファイルに正しい値を埋め込みます。また，リンカは，リンク時の情報をリンク・マップ・ファイルとして出力します。

図 1-7 リンカの機能



注 デバイス・ファイルはオンライン・デリバリ・サービス（ODS）から別途入手してください。
下記 URL から取得可能です。

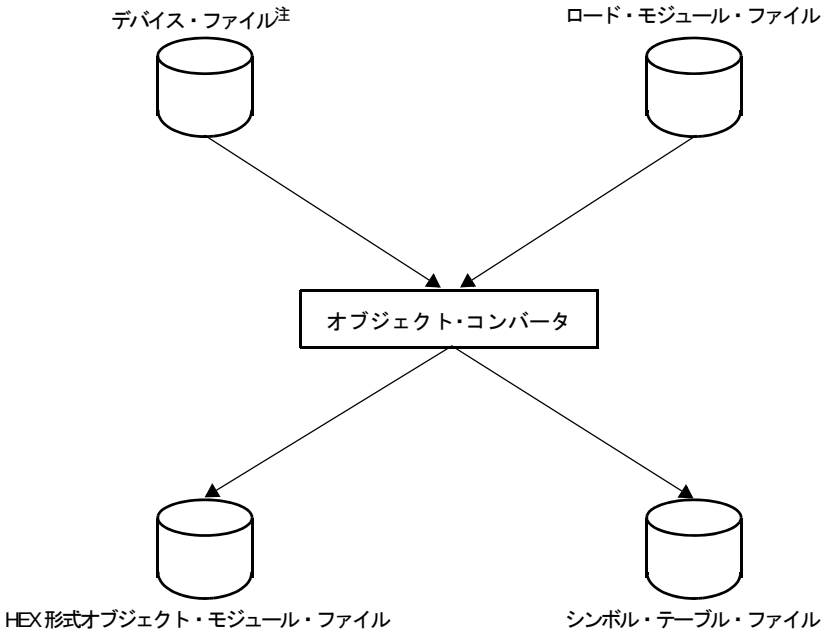
http://www.necel.com/micro/ods/jpn/index.html

1.2.5 オブジェクト・コンバータ

コンバートは，RA78K0R　アセンブラ・パッケージ（別売）に含まれているオブジェクト・コンバータを用いて行います。

オブジェクト・コンバータは，リンカの出力したロード・モジュール・ファイルを入力し，インテル標準 HEX 形式オブジェクト・モジュール・ファイルへと変換します。また，オブジェクト・コンバータは，コンバート時の情報をシンボル・テーブル・ファイルとして出力します。

図 1-8 オブジェクト・コンバータの機能



注 デバイス・ファイルはオンライン・デリバリ・サービス（ODS）から別途入手してください。
下記 URL から取得可能です。

http://www.necel.com/micro/ods/jpn/index.html

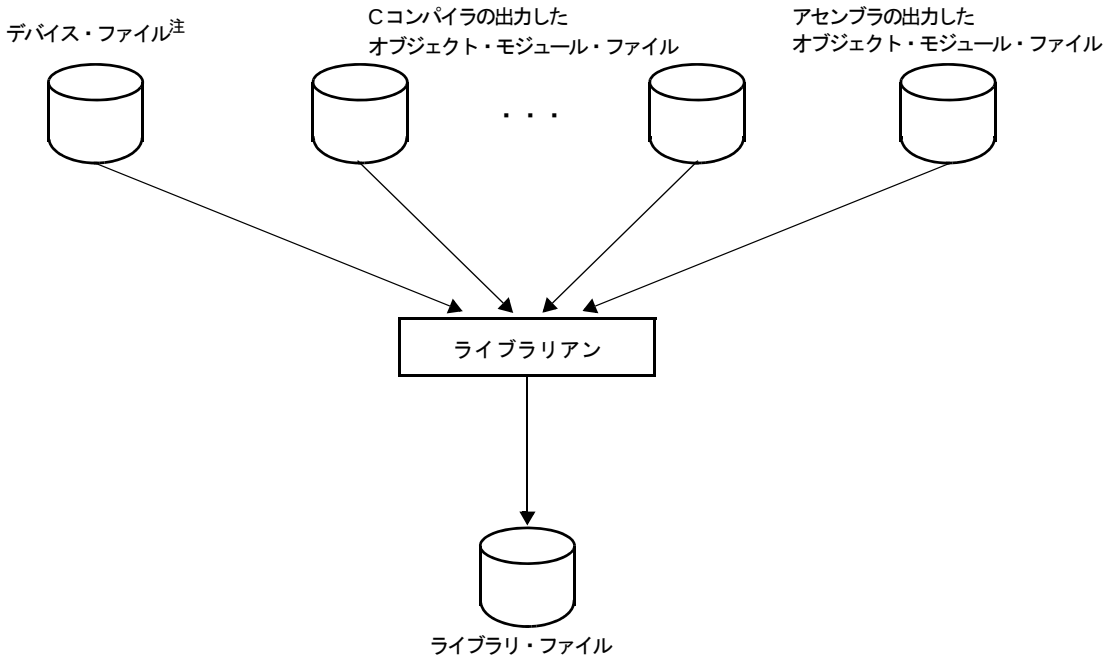
1.2.6 ライブラリアン

汎用性のある，インタフェースの明確なモジュールは，ライブラリ化しておくと便利です。ライブラリ化することにより，多くのオブジェクト・モジュールも 1 つのファイルになり，扱いやすくなります。

リンカには，ライブラリ・ファイルの中から必要なモジュールだけを取り出してリンクする機能があります。したがって，複数のモジュールを 1 つのライブラリ・ファイルに登録しておけば，リンク時に必要なモジュール・ファイル名をいちいち指定する必要はなくなります。

ライブラリアンは，RA78K0R　アセンブラ・パッケージ（別売）に含まれているライブラリアンを用いて行います。

図 1-9 ライブラリアンの機能



注 デバイス・ファイルはオンライン・デリバリ・サービス（ODS）から別途入手してください。

下記 URL から取得可能です。

http://www.necel.com/micro/ods/jpn/index.html

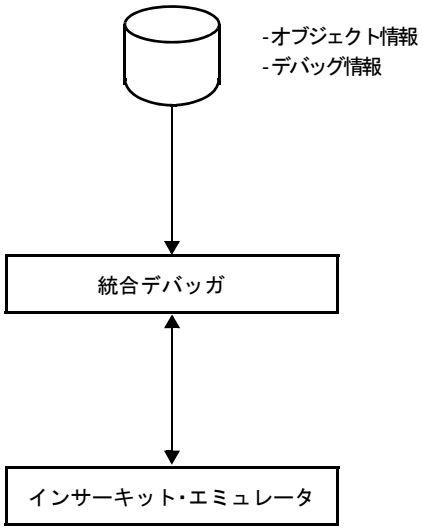
1.2.7 デバッガ

リンカが出力したロード・モジュール・ファイルを ID78K0R-QB（78K0R シリーズ用統合デバッガ）を使用して，IE（インサーキット・エミュレータ）にロードすることにより，GUI を用いたソース・デバッグが可能となります。

デバッグのために，対象のソース・プログラムをコンパイルする際，デバッグ情報出力指定オプション -g を指定しておきます（-g はデフォルト・オプションです）。その指定により，デバッグに必要なシンボルと行番号の情報がオブジェクト・モジュール内に付加されます（コンパイラ・オプションについては，「第 5 章 コンパイラ・オプション」を参照してください）。

デバッガ，IE は，別パッケージ（別売）になります。

図 1-10 デバッガの機能



1.2.8 システム・シミュレータ

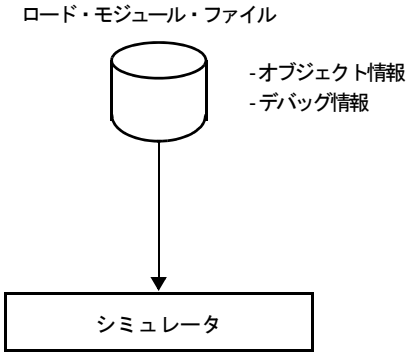
リンカが出力したロード・モジュール・ファイルを SM+ for 78K0R（78K0R シリーズ用システム・シミュレータ）でダウンロードすることにより，GUI を用いたソース・デバッグが可能となります。

SM+ for 78K0R は，ID78K0R-QB と同様な操作イメージで，ロード・モジュール・ファイルをホスト・マシン上でシミュレーションするためのソフトウエアです。

SM+ for 78K0R では，機械語命令のシミュレーションに加え，デバイス内蔵の周辺や割り込みもシミュレーション可能です。疑似的なターゲット・システムを構築するための外部部品や手段を提供しているので，ターゲット・システムの動作を含めたプログラムのデバッグを，ハードウエア開発から独立して早期に行うことができます。

システム・シミュレータは，別パッケージ（別売）になります。

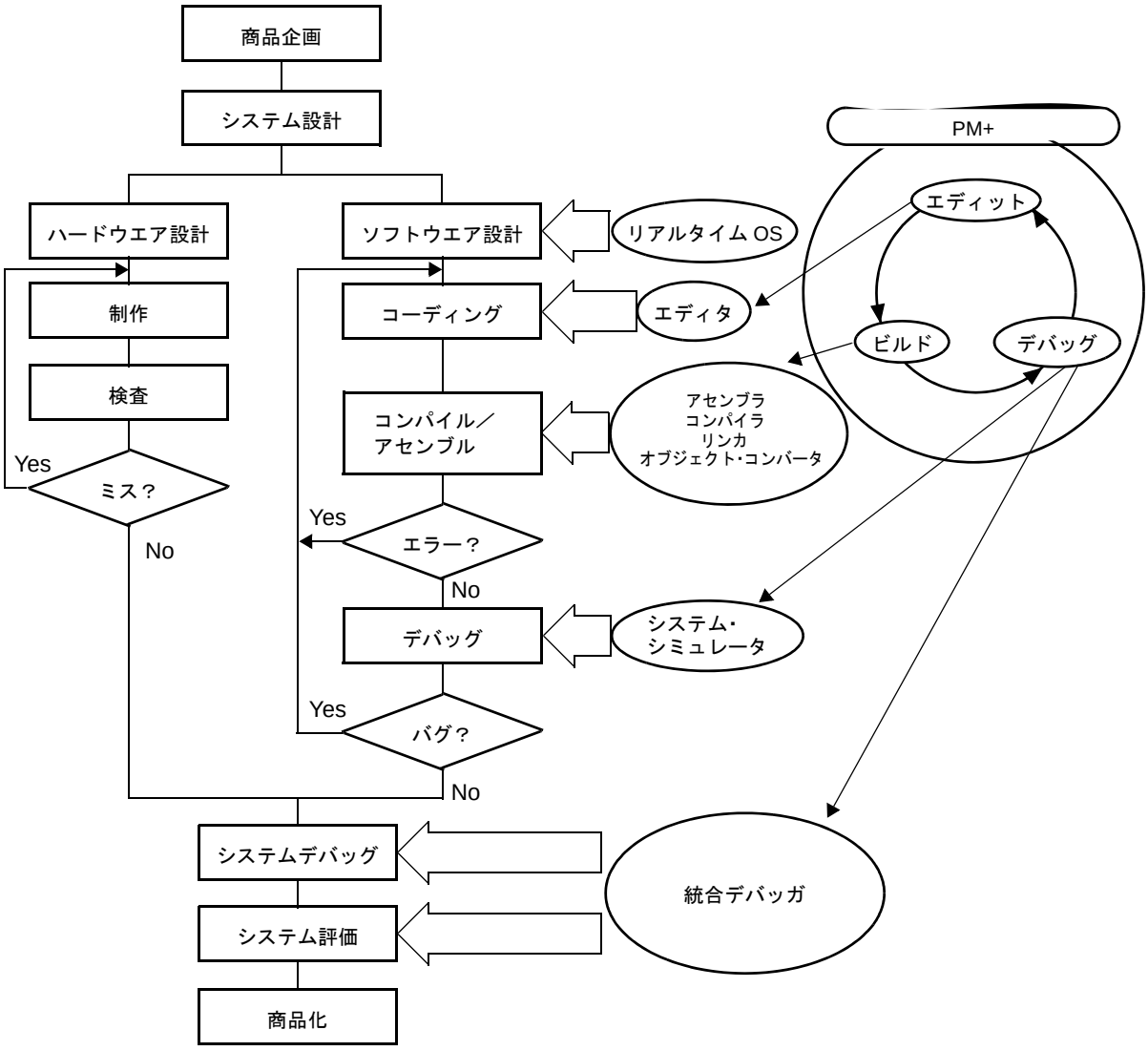
図 1-11 シミュレータの機能



1.2.9 PM+

PM+ は，ユーザ・プログラムを効率よく開発するための統合開発環境プラットホームであり，エディタ，ビルダ，ディバッガを起動するなど，ユーザ・プログラムの開発における一連の作業を行うことができます。

図 1-12 PM+ の機能



第 2 章　製品概要とインストール方法

この章では，CC78K0R の提供媒体に格納されているファイル群をユーザの開発環境（ホスト・マシン）上にインストールする際の手順，およびユーザの開発環境上からアンインストールする際の手順について説明します。

2.1　ホスト・マシンと提供媒体

CC78K0R は，次に示す開発環境に対応しています。

表 2-1　CC78K0R の提供媒体と記録形式

ホスト・マシン	OS	提供媒体
IBM PC/AT™ 互換機	Windows（2000/XP）注	CD-ROM

注　CC78K0R を Windows 上で使用するためには，PM+ が必要です。PM+ を使用しない場合は，コマンド・プロンプトで CC78K0R を起動することが可能です。

2.2 インストール

以下に，CC78K0R の提供媒体に格納されているファイル群をホスト・マシン上にインストールする際の手順を示します。

1. Windows の起動

   ホスト・マシン，および周辺機器などの電源を投入し，Windows を起動します。

2. 提供媒体のセット

   CC78K0R の提供媒体を，ホスト・マシンの該当デバイス装置（CD-ROM ドライブ）にセットすることにより，セットアップ・プログラムが自動実行します。
   以降，モニタ画面に表示されるメッセージに従って，インストール作業を実行します。

   注意 セットアップ・プログラムが自動実行しない場合には，ルートの INSTALL.EXE を起動します。

3. ファイル群の確認

   Windows の標準アプリケーション“エクスプローラ”などを用いて，CC78K0R の提供媒体に格納されていたファイル群がホスト・マシン上にインストールされたことを確認します。
   なお，各フォルダについての詳細は「2.4 フォルダ構成」を参照してください。

2.3 デバイス・ファイルのインストール

デバイス・ファイルはオンライン・デリバリ・サービス（ODS）から別途入手してください。

下記 URL から取得可能です。

http://www.necel.com/micro/ods/jpn/index.html

デバイス・ファイルのインストールには，“デバイスファイルインストーラ”を使用します。“デバイスファイルインストーラ”は，CC78K0R とともにインストールされます。

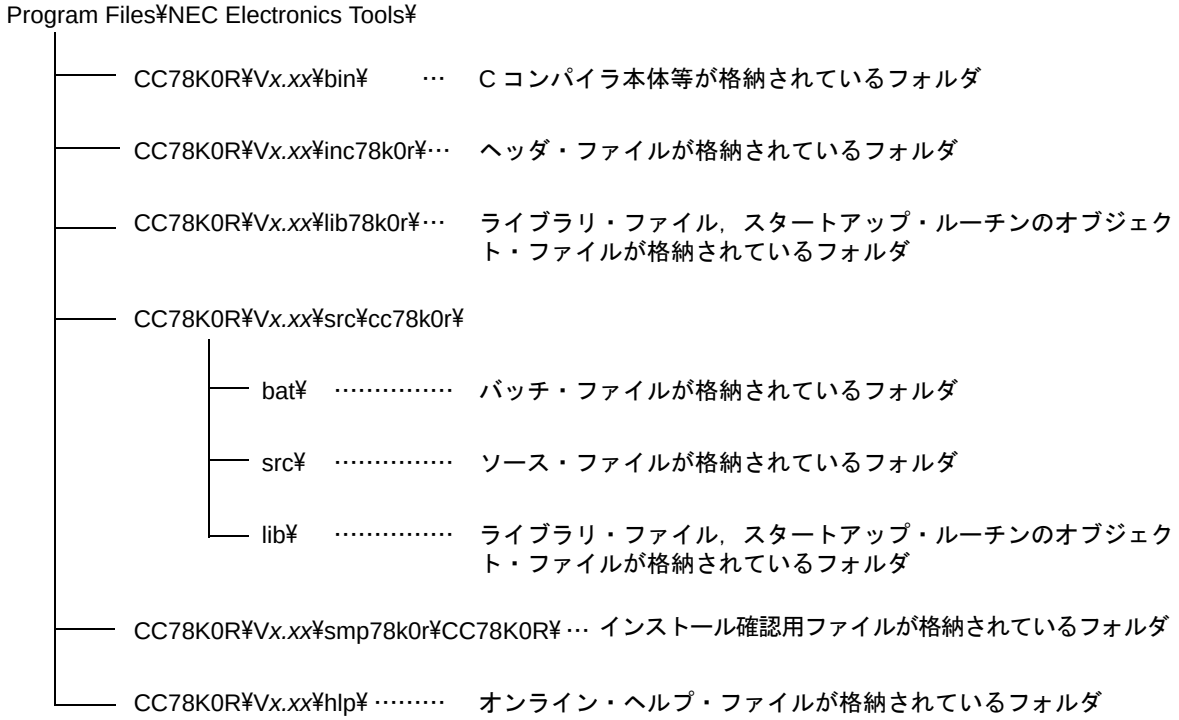
2.4 フォルダ構成

インストール時に表示される標準フォルダは，Windows システムの“Program Files¥NEC Electronics Tools”です。インストール・フォルダ下の構成は，次のようになります。ただし，ドライブやインストール・フォルダはインストール時に変更してもかまいません。

PM+ によるメイクを行う場合は，各ツール（CC78K0R，RA78K0R など）を同じドライブ，およびフォルダにインストールしてください。

なお，このマニュアルでは，セットアップ・プログラムのデフォルトの指示に従って，デフォルトのプログラム名“Program Files¥NEC Electronics Tools”で標準フォルダにインストールしたこととして説明しています。

図 2-1 フォルダ構成



2.5 ファイル構成

各フォルダの内容を，次に示します。

フォルダ構造，およびファイル構成は，インストーラ使用時のものです。

表 2-2 ファイル構成

フォルダ名	ファイル名	概要
CC78K0R¥V*x.xx*¥bin¥	cc78k0r.exe	C コンパイラ本体
	cc78k0r.msg	メッセージ・ファイル
	*.hlp	ヘルプ・メッセージ用ファイル
	*.dll	DLL ファイル
CC78K0R¥V*x.xx*¥inc78k0r¥	*.h 注 1	標準ライブラリのヘッダ・ファイル
CC78K0R¥V*x.xx*¥lib78k0r¥（リンク用）注 2，3	cl0r*.lib	ライブラリ・ファイル（ランタイム・ライブラリ，標準ライブラリ）
	s0r*.rel	スタートアップ・ルーチンのオブジェクト・ファイル
CC78K0R¥V*x.xx*¥src¥cc78k0r¥bat¥ 注 4	mkstup.bat	スタートアップ・ルーチンのアセンブル用バッチ・ファイル
	reprom.bat	rom.asm 更新用バッチ・ファイル
	*.bat 注 5	標準関数（一部）などの更新用バッチ・ファイル
CC78K0R¥V*x.xx*¥src¥cc78k0r¥src¥	cstart*.asm 注 6	スタートアップ・ルーチンのソース・ファイル
	rom.asm	ROM 化ルーチンのソース・ファイル
	*.asm 注 7	標準関数（一部）などのソース・ファイル
CC78K0R¥V*x.xx*¥lib78k0r¥lib¥（変更用）注 2	cl0r*.lib	ライブラリ・ファイル（ランタイム・ライブラリ，標準ライブラリ）
	s0r*.rel	スタートアップ・ルーチンのオブジェクト・ファイル
CC78K0R¥V*x.xx*¥smp78k0r¥CC78K0R¥	prime.c	インストール確認用ソース・プログラム
	sample.bat	インストール確認用バッチ・ファイル
	readme.doc	インストール確認用ファイルの説明
	lk78k0r.dr	参考用リンク・ディレクティブ・ファイル
CC78K0R¥V*x.xx*¥hlp¥	cc78k0rp.chm	オンライン・ヘルプ・ファイル

備考 * ： 英数字

注 1 「CC78K0R C コンパイラ 言語編」のユーザーズ・マニュアル を参照してください。

注 2 スタートアップ・ルーチンを修正する場合は，CC78K0R¥V*x.xx*¥src¥cc78k0r¥lib フォルダ以下のソース・ファイルを修正してください。
バッチ・ファイルでアセンブルしたものが CC78K0R¥V*x.xx*¥src¥cc78k0r¥lib フォルダに入るので，そのファイルを CC78K0R¥V*x.xx*¥lib78k0r フォルダにコピーして，ユーザ・プログラムとリンクしてください。

注 3 「2.5.1 ライブラリ・ファイル」を参照してください。

注 4 このフォルダにあるバッチ・ファイルは，PM+ 上では使用することができません。バッチ・ファイルを使用する場合は，コマンド・プロンプトで実行してください。また，これらのバッチ・ファイルは，ソース修正が必要な場合のみ，使用してください。

注 5 表 8-1 を参照してください。

注 6 * = B | E | N （B：ブート領域指定時，E：フラッシュ領域指定時，N：標準ライブラリ未使用）

注 7 表 8-2 を参照してください。

2.5.1 ライブラリ・ファイル

ライブラリ・ファイルは，標準ライブラリ，ランタイム・ライブラリ，およびスタートアップ・ルーチンで構成されます。

フォルダの内容を，次に示します。

表 2-3 ライブラリ・ファイル

フォルダ名	ファイル名			ファイルの役割
	通常	ブート領域	フラッシュ領域	
lib78k0r¥	cl0rm.lib cl0rl.lib cl0rmf.lib cl0rlf.lib cl0rxm.lib 注 3 cl0rxl.lib 注 3	cl0rm.lib cl0rl.lib cl0rmf.lib cl0rlf.lib cl0rxm.lib 注 3 cl0rxl.lib 注 3	cl0rme.lib cl0rle.lib cl0rmfe.lib cl0rlfe.lib cl0rxme.lib 注 3 cl0rxle.lib 注 3	ライブラリ・ファイル（ランタイム・ライブラリ，標準ライブラリ）注 1
	s0rm.rel s0rml.rel s0rl.rel s0rll.rel	s0rmb.rel s0rmlb.rel s0rlb.rel s0rllb.rel	s0rme.rel s0rmle.rel s0rle.rel s0rlle.rel	スタートアップ・ルーチンのオブジェクト・ファイル注 2

注 1 ライブラリ・ファイルの命名規則は，次のようになっています。

```
lib78k0r¥cl0r<mul><model><float><flash>.lib
```

<mul>

なし ： 乗算器未使用

x ： 乗算器使用

<model>

m ： スモール・モデル，ミディアム・モデル

l ： コンパクト・モデル，ラージ・モデル

<float>

なし ： 標準ライブラリ，ランタイム・ライブラリ（浮動小数点ライブラリ未使用）

f ： 浮動小数点ライブラリ用

<flash>

なし ： 通常／ブート領域用

e ： フラッシュ領域用

注 2 スタートアップ・ルーチンの命名規則は，次のようになっています。

```
lib78k0r¥s0r<model><lib><flash>.rel
```

<*model*>

m　：ミディアム・モデル（スモール・モデル兼用）

l　：ラージ・モデル（コンパクト・モデル兼用）

<*lib*>

なし：標準ライブラリ関数を使用しない場合

l　：標準ライブラリ関数を使用する場合

<*flash*>

なし：通常用

b　：ブート領域用

e　：フラッシュ領域用

注 3 CC78K0R のライブラリでは，次のようなデバイスの乗算器に対応しています。
ただし，演算途中に割り込みが入った場合に，演算結果を壊さないために，割り込み禁止にしている部分があります。
対象となるライブラリ関数と，割り込み禁止時間については，「CC78K0R C コンパイラ 言語編」のユーザーズ・マニュアルを参照してください。

[特殊機能レジスタ]

機能	予約語	アドレス	サイズ
乗算入力データ A	MULA	FFFF0H	16 ビット
乗算入力データ B	MULB	FFFF2H	16 ビット
乗算結果データ	MULOH，MULOL	FFFF4H，FFF6H	16 ビット× 2

[レジスタ構成]

＜乗数 A ＞　　＜乗数 B ＞　　＜積＞

MULA（ビット 15-0）× MULB（ビット 15-0）=MULOH（上位）（ビット 15-0），MULOL（下位）（ビット 15-0）

2.6 アンインストール

以下に，ホスト・マシンにインストールされているファイル群をアンインストールする際の手順を示します。

1. Windows の起動

   ホスト・マシン，および周辺機器などの電源を投入し，Windows を起動します。

2. CC78K0R の削除

   コントロール・パネルの [アプリケーションの追加と削除]，または [プログラムの追加と削除] において，“NEC EL CC78K0R V*x.xx*”を選択してください。

3. ファイル群の確認

   Windows の標準アプリケーション“エクスプローラ”などを用いて，ホスト・マシンにインストールされていたファイル群がアンインストールされたことを確認してください。
   なお，各フォルダについての詳細は，「2.4 フォルダ構成」を参照してください。

2.7 環境設定

2.7.1 ホスト・マシン

CC78K0R は，32 ビット化されており，i386™ 以上の CPU を搭載した機種で動作します。

- Windows 2000/XP
- Windows 2000/XP のコマンド・プロンプト

2.7.2 環境変数

コマンド・プロンプトでの作業時には，次の環境変数を設定してください。

表 2-4 環境変数

環境変数	説明
PATH	C コンパイラを置いたフォルダを指定します。
TMP	一時ファイルを作成するフォルダを指定します。
LANG78K	ソース・ファイル中の漢字コード（2 バイト・コード）を指定します。 sjis ： シフト JIS（デフォルト） euc ： EUC none ： 漢字コードなし
INC78K0R	C コンパイラの標準ヘッダ・ファイルを置いたフォルダを指定します。
LIB78K0R	C コンパイラのライブラリを置いたフォルダを指定します。

【指定例】

```
PATH=%PATH%;C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥bin
set TMP=C:¥tmp
set LANG78K=sjis
set INC78K0R=C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥inc78k0r
set LIB78K0R=C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r
```

第 3 章　コンパイルからリンクまでの手順

この章では，CC78K0R と RA78K0R アセンブラ・パッケージを使用し，コンパイルからリンクまでを行う手順を説明します。

この章の実行手順に従って，実際にサンプル・プログラム“prime.c”をコンパイルからリンクまでを行うことにより，コンパイル，アセンブル，リンクの操作に慣れることができます（サンプル・プログラムについては，「付録 A サンプル・プログラム」を参照してください）。

ここでは，PM+ 上で実行する方法と，コマンド・ラインで実行する方法を説明します（インストールについては，「2.2 インストール」を参照してください）。

3.1　PM+ について

ここでは，RA78K0R アセンブラ・パッケージに含まれる PM+ で CC78K0R を起動する場合のユーザ・インタフェースについて説明します。

PM+ から CC78K0R を起動する場合，CC78K0R に含まれる cc78k0rp.dll が参照されます。

3.1.1　cc78k0rp.dll（ツール DLL）の位置づけ

cc78k0rp.dll ファイルなどのツール DLL ファイルは，PM+ から使用するために必要なファイルです。

3.1.2　実行環境

PM+ に準じます。

3.1.3 CC78K0R 用オプション設定メニュー

(1) オプション・メニュー項目

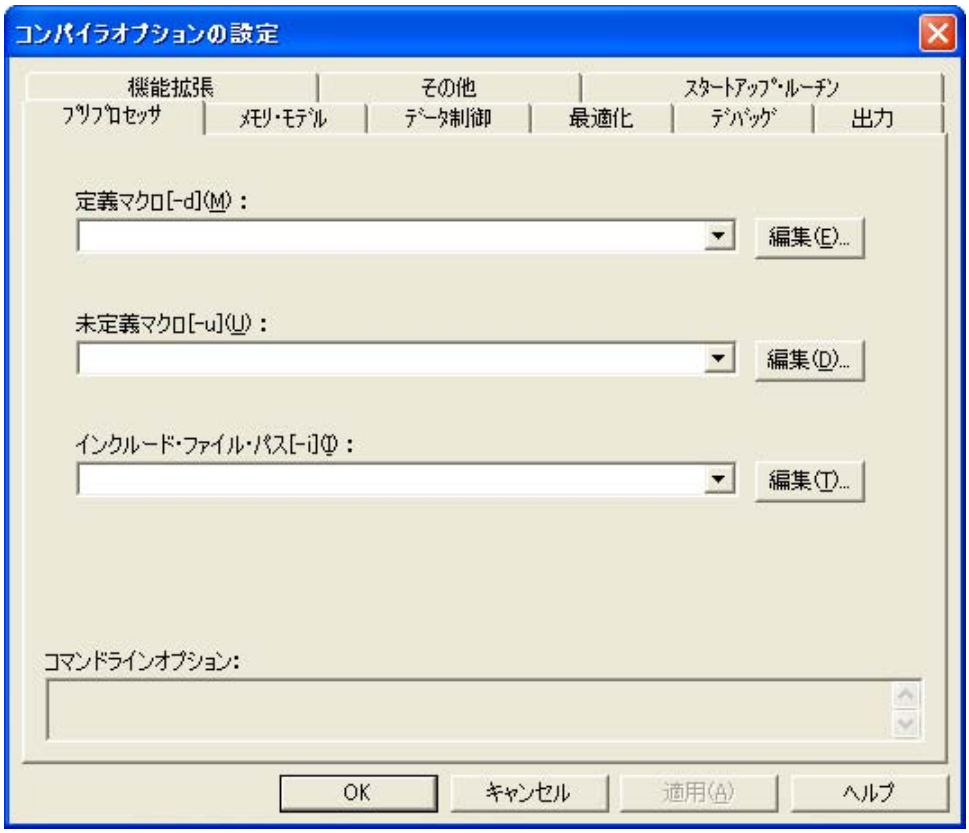
CC78K0R C コンパイラ・パッケージに含まれるツール DLL ファイルにより，PM+ の [ツール (T)] メニュー中に次の項目が追加されます。

[コンパイラオプションの設定 (C)]

(2) [コンパイラオプションの設定] ダイアログ

PM+ の [ツール (T)] メニュー→ [コンパイラオプションの設定 (C)] を選択すると，ツール DLL のオプション設定機能が呼び出され，[コンパイラオプションの設定] ダイアログがオープンします。

図 3-1 [コンパイラオプションの設定] ダイアログ



(a) [フォルダの参照] ダイアログ

[コンパイラオプションの設定] ダイアログで，以下のパス設定に対して [参照] ボタンを押すと，[フォルダの参照] ダイアログがオープンします。

なお，このダイアログでは，フォルダのみを指定することができます。

- [出力] タブのオブジェクト・モジュール・ファイルの出力パス
- [出力] タブのアセンブラ・モジュール・ファイルの出力パス
- [出力] タブのエラー・リスト・ファイルの出力パス
- [出力] タブのクロスリファレンス・リスト・ファイルの出力パス
- [出力] タブのプリプロセス・リスト・ファイルの出力パス
- [その他] タブのテンポラリ・ファイル・パス

図 3-2 [フォルダの参照] ダイアログ

(b) [ParameterFile] ダイアログ

[コンパイラオプションの設定] ダイアログで，以下のパス設定に対して [参照] ボタンを押すと，[ParameterFile] ダイアログがオープンします。

- [その他] タブのパラメータ・ファイル

このダイアログのデフォルトは次のとおりです。

カレント・フォルダ　：プロジェクト・ファイルのフォルダ
ファイルの種類　　：パラメータ・ファイル（*.pcc）

図 3-3 [ParameterFile] ダイアログ

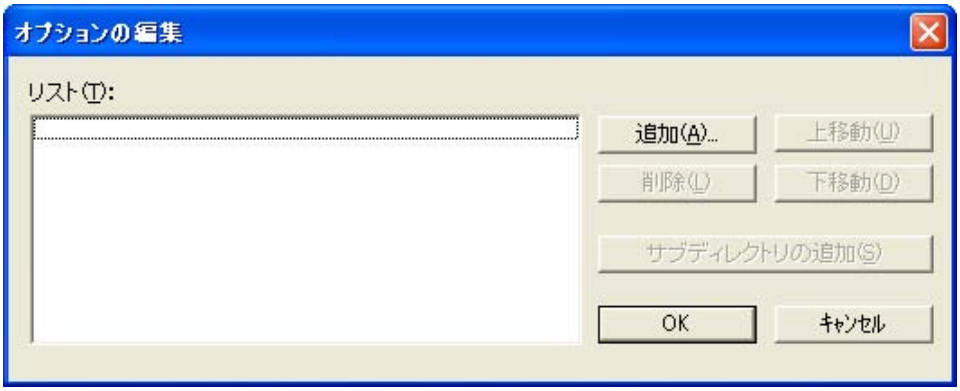
(c) [オプションの編集] ダイアログ

[コンパイラオプションの設定] ダイアログで，以下の設定に対して [編集] ボタンを押すと，[オプションの編集] ダイアログがオープンします。

- [プリプロセッサ] タブの定義マクロ
- [プリプロセッサ] タブの未定義マクロ
- [プリプロセッサ] タブのインクルード・ファイル・パス

[オプションの編集] ダイアログは，項目をリストで編集します。

図 3-4 [オプションの編集] ダイアログ



[オプションの編集] ダイアログについて，次に説明します。

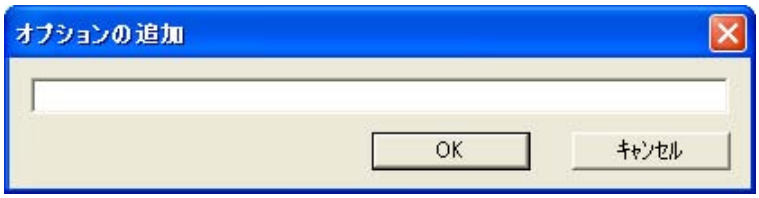
- [追加 (A)...] ボタン

  リストの項目を追加します。

  ファイルやフォルダを指定する項目の場合は，それぞれの [フォルダの参照] ダイアログがオープンします。

  それ以外の場合は，内容を入力する [オプションの追加] ダイアログがオープンします。

図 3-5 [オプションの追加] ダイアログ



- [削除 (L)] ボタン

  選択中のリストの項目を削除します。

- [上移動 (U)] ボタン

  選択中のリストの項目を上に移動します。

- [下移動 (D)] ボタン

  選択中のリストの項目を下に移動します。

- [サブディレクトリの追加 (S)] ボタン

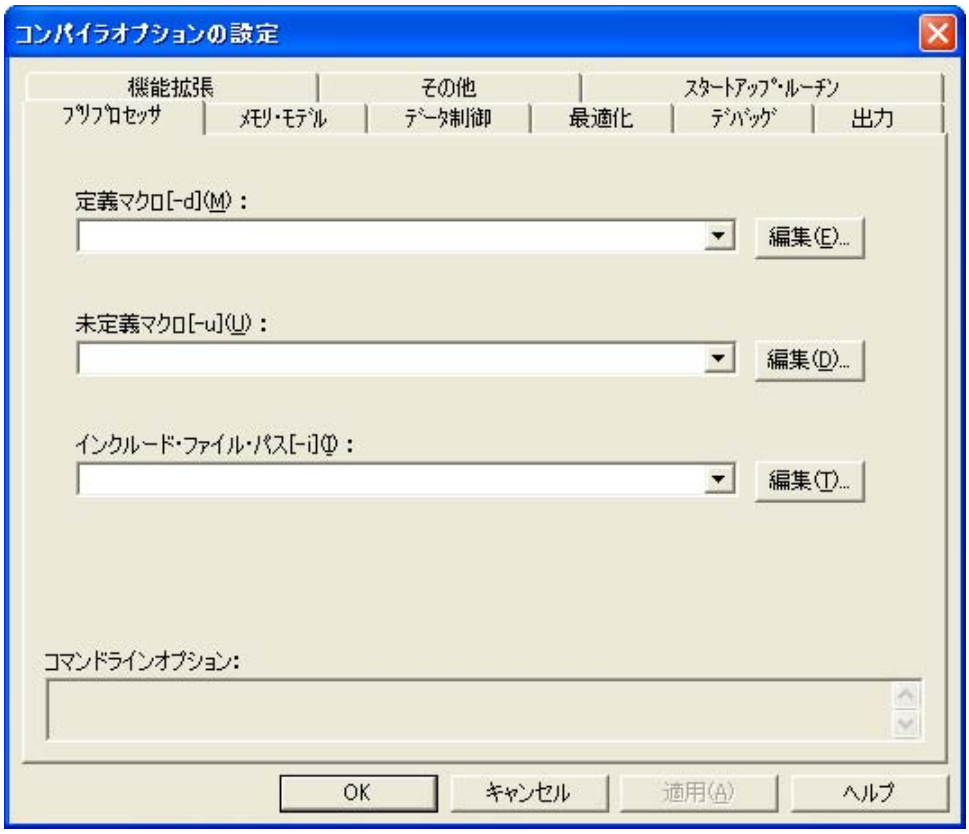
次の場合は，選択中のリストの項目にサブディレクトリを追加することができます。

[プリプロセッサ] タブの“インクルード・ファイル・パス [-i](I)”を編集する場合

3.1.4 [コンパイラオプションの設定] ダイアログの各部の説明

ここでは，[コンパイラオプションの設定] ダイアログの各部について説明します。

図 3-6 [コンパイラオプションの設定] ダイアログ



- コンパイラオプションの設定

各コンパイラ・オプションを次の 9 種類に分けて設定します。

ダイアログ上部のタブをクリックすることによって，それぞれの設定画面に切り替わります。

[プリプロセッサ] タブ（デフォルト）

[メモリ・モデル] タブ

[データ制御] タブ

[最適化] タブ

[デバッグ] タブ

[出力] タブ

[機能拡張] タブ

[その他] タブ

[スタートアップ・ルーチン] タブ

- コマンドラインオプション

現在指定されているオプション文字列を表示します。

指定されたオプション文字列は，リアルタイムに反映し，表示します。

この表示領域には，入力することはできません。

CC78K0R のデフォルト・オプションについては，最初から指定されている状態（チェック・ボックス等にチェックされた状態）になっていますが，“コマンドラインオプション”には表示されません。

オプション文字列表示領域に収まりきらないオプションは，スクロールバーでスクロールすることにより確認することができます。

- [OK] ボタン

このダイアログで編集した設定を決定し，[コンパイラオプションの設定] ダイアログを閉じます。

プロジェクト・ウインドウで個別コンパイラ・オプションの設定を選択した場合はそのソース・ファイルに対して，[ツール (T)] メニューでコンパイラ・オプションの設定を選択した場合は全体のソース・ファイルに対して，オプション設定が行われます。

- [キャンセル] ボタン

オプション設定を行わず，ダイアログを閉じます。

ESC キーは，フォーカスがダイアログのどこにあっても，[キャンセル] ボタンと同等の効果を持ちます。

- [適用 (A)] ボタン

このボタンは，オプション設定を変更した場合のみ有効となります。

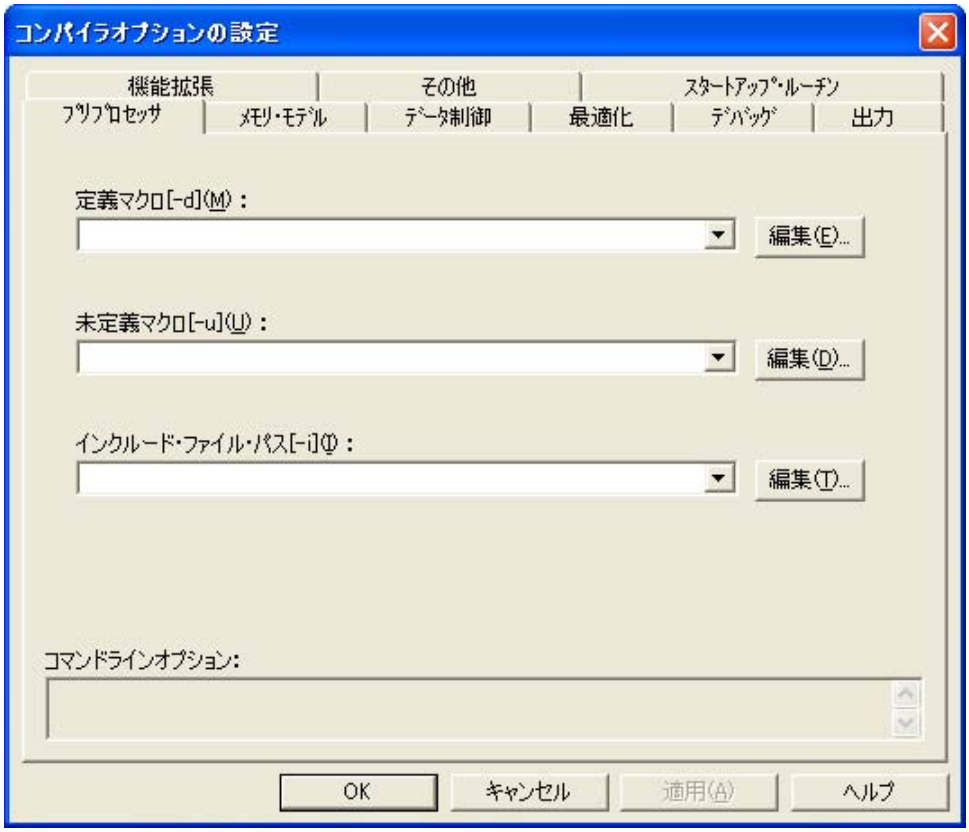
このダイアログで編集した設定を決定し，[コンパイラオプションの設定] ダイアログを開いたままにします。

- [ヘルプ] ボタン

このダイアログに関するヘルプ・メッセージを表示します。

[プリプロセッサ] タブ

図 3-7 [コンパイラオプションの設定] ダイアログ（[プリプロセッサ] タブ選択時）



- 定義マクロ [-d](M)

-d で指定するマクロ名，および定義名を，コンボ・ボックスに入力します。

マクロ名は，“ , ”で区切ることにより，一度に 30 個までマクロ定義を行うことができます。

1 つの定義マクロの指定可能な文字数は，256 文字です。

コンボ・ボックスに入力可能な文字数は，7,709 文字です。

[編集 (E)...] ボタンから指定することもできます（[オプションの編集] ダイアログをオープンします）。

定義マクロが重複した場合は，エラー・メッセージが表示されます。

- 未定義マクロ [-u](U)

-u で指定するマクロ名を，コンボ・ボックスに入力します。

マクロ名は，“ , ”で区切ることにより，一度に 30 個までマクロ定義を無効にすることができます。

1 つの未定義マクロの指定可能な文字数は，256 文字です。

コンボ・ボックスに入力可能な文字数は，7,709 文字です。

[編集 (D)...] ボタンから指定することもできます（[オプションの編集] ダイアログをオープンします）。

未定義マクロが重複した場合は，エラー・メッセージが表示されます。

- インクルード・ファイル・パス [-i](I)

-i で指定するインクルード・ファイルがあるフォルダを，コンボ・ボックスに入力します。

”,”で区切ることにより，一度に 64 個までフォルダを指定することができます。

1 つのインクルード・ファイル・パスの指定可能な文字数は，259 文字です。

コンボ・ボックスに入力可能な文字数は，16,639 文字です。

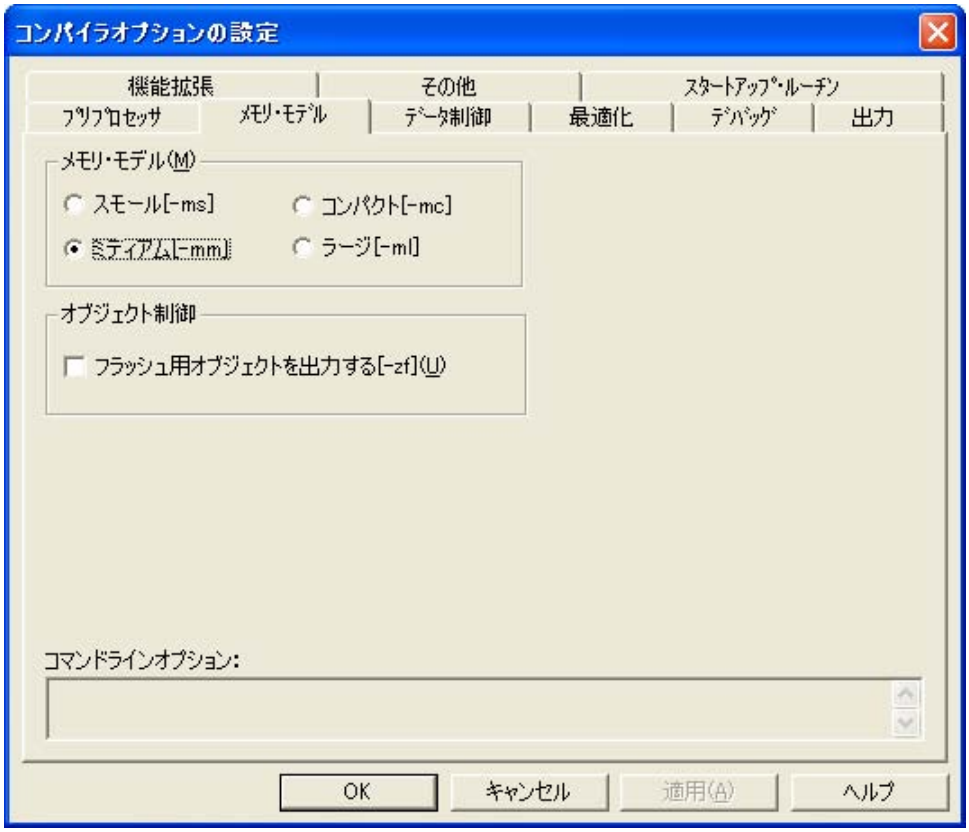
[編集 (T)...] ボタンから指定することもできます（[オプションの編集] ダイアログをオープンします）。

存在しないパスは，指定することができません。

パスが重複した場合は，エラー・メッセージが表示されます。

[メモリ・モデル] タブ

図 3-8 [コンパイラオプションの設定] ダイアログ（[メモリ・モデル] タブ選択時）



注意 ソース・ファイル単位で個別コンパイラ・オプションを設定する際は，[メモリ・モデル] タブの設定を行うことはできません。

- メモリ・モデル (M)

コンパイル時のメモリ・モデルの種類を，ラジオ・ボタンで選択します。

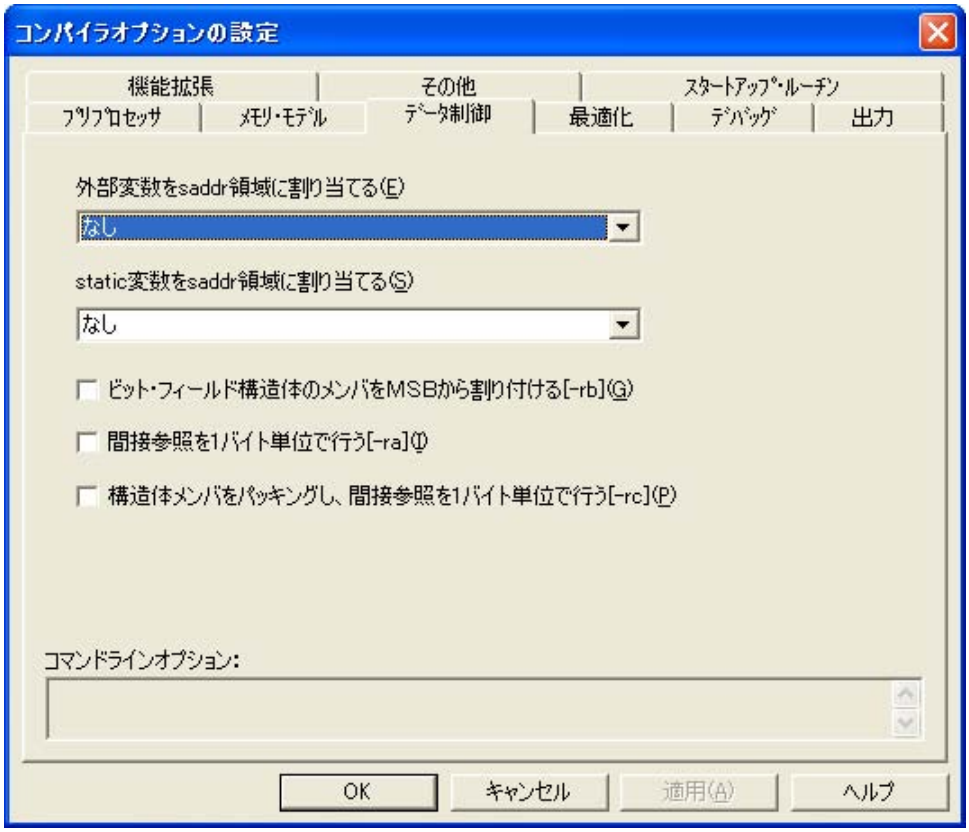
- オブジェクト制御

フラッシュ用オブジェクトを出力する [-zf](U)

-zf オプションを有効にする場合にチェック・ボックスをチェックします。

[データ制御] タブ

図 3-9 [コンパイラオプションの設定] ダイアログ（[データ制御] タブ選択時）



- 外部変数を saddr 領域に割り当てる (E)

saddr 領域に配置させる外部変数の種類を，ドロップダウン・リストから選択します。

注意 ソース・ファイル単位で個別コンパイラ・オプションを設定する際は，本項目の設定を行うことはできません。

- static 変数を saddr 領域に割り当てる (S)

saddr 領域に配置させる static 変数の種類を，ドロップダウン・リストから選択します。

- ビット・フィールド構造体のメンバを MSB から割り当てる [-rb](G)

-rb オプションを有効にする場合に，チェック・ボックスをチェックします。

- 間接参照を 1 バイト単位で行う [-ra](I)

-ra オプションを有効にする場合にチェック・ボックスをチェックします。

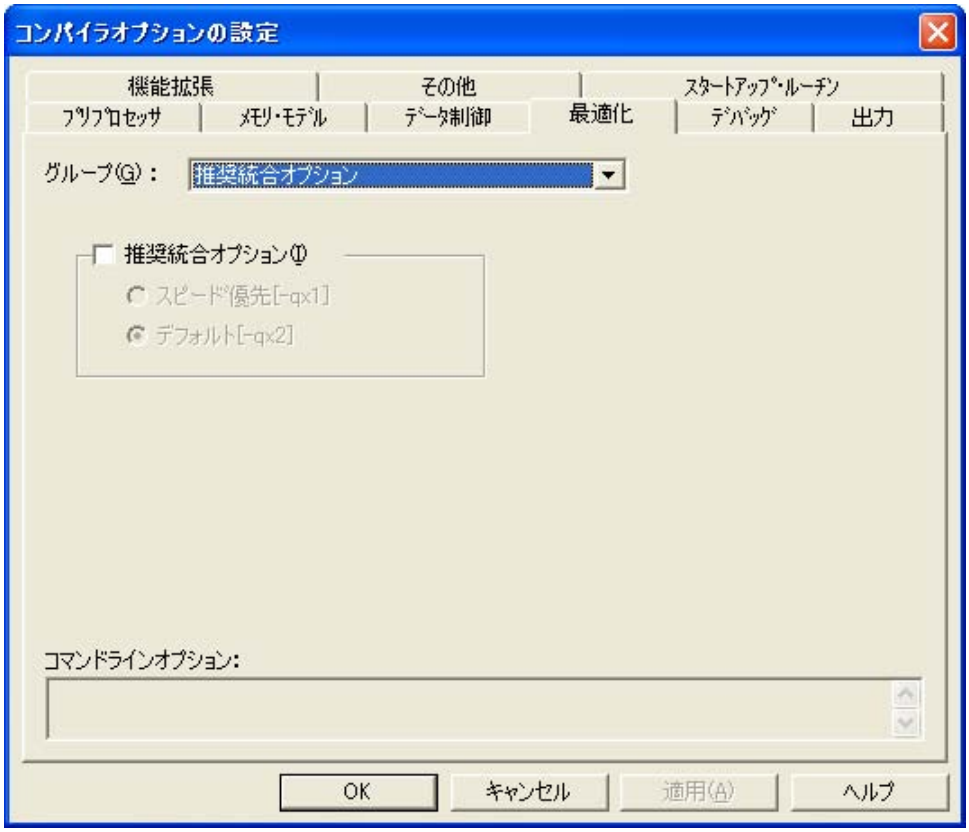
- 構造体メンバをパッキングし、間接参照を 1 バイト単位で行う [-rc](P)

-rc オプションを有効にする場合にチェック・ボックスをチェックします。

[最適化] タブ

(1)“グループ (G)”で“推奨統合オプション”を選択した場合

図 3-10 [コンパイラオプションの設定] ダイアログ（“推奨統合オプション”選択時）



- 推奨統合オプション (I)

推奨統合オプションとは，最適化オプションを個々に指定する代わりに，目的別に統合して最適化オプションをより使いやすくしたものです。

スピード優先，デフォルトの 2 パターンがあり，それぞれの意味は次の通りです。

スピード優先 [-qx1]

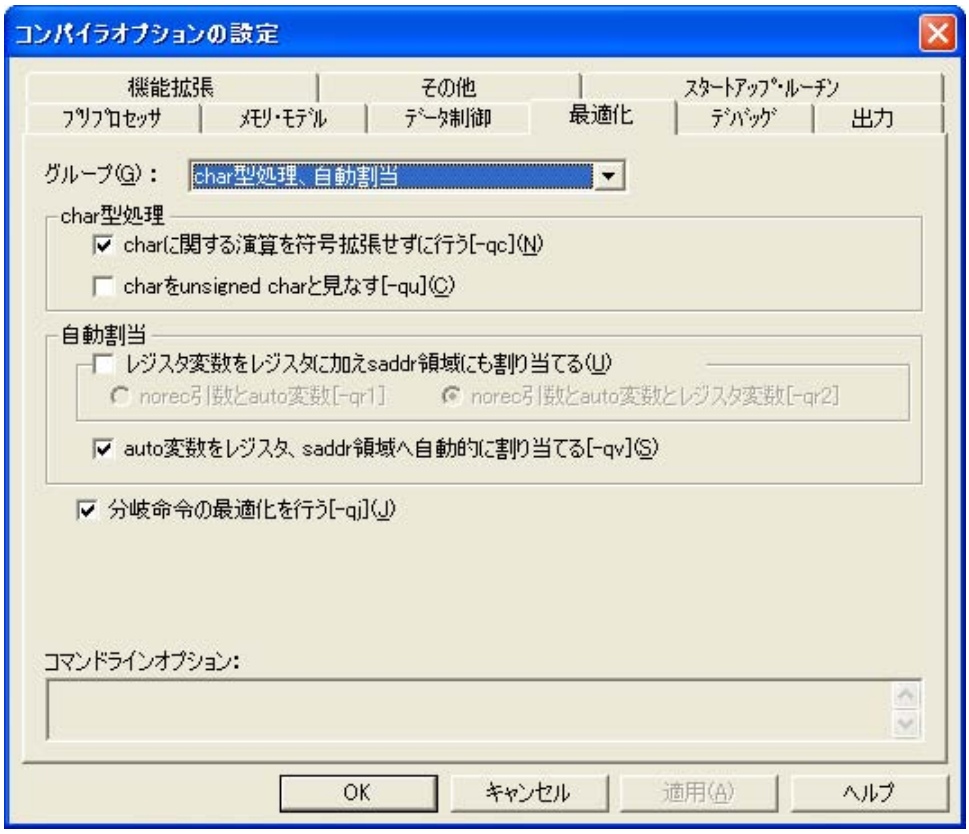
特に実行スピード効率を重視する場合に選択します。

デフォルト [-qx2]

実行スピード，オブジェクト・コード・サイズ効率の両方とも重視する場合に選択します。

(2) “グループ (G)” で “char 型処理、自動割当” を選択した場合

図 3-11 [コンパイラオプションの設定] ダイアログ（“char 型処理、自動割当” 選択時）



- char 型処理

char に関する演算を符号拡張せずに行う [-qc](N)

-qc オプション（汎整数拡張をしない）を有効にする場合に，チェック・ボックスをチェックします。

char を unsigned char と見なす [-qu](C)

-qu オプションを有効にする場合に，チェック・ボックスをチェックします。

- 自動割当

レジスタ変数をレジスタに加え saddr 領域にも割り当てる (U)

-qr オプションを有効にする場合にチェック・ボックスをチェックし，割り当てる変数をラジオ・ボタンで選択します。

auto 変数をレジスタ、saddr 領域へ自動的に割り当てる [-qv](S)

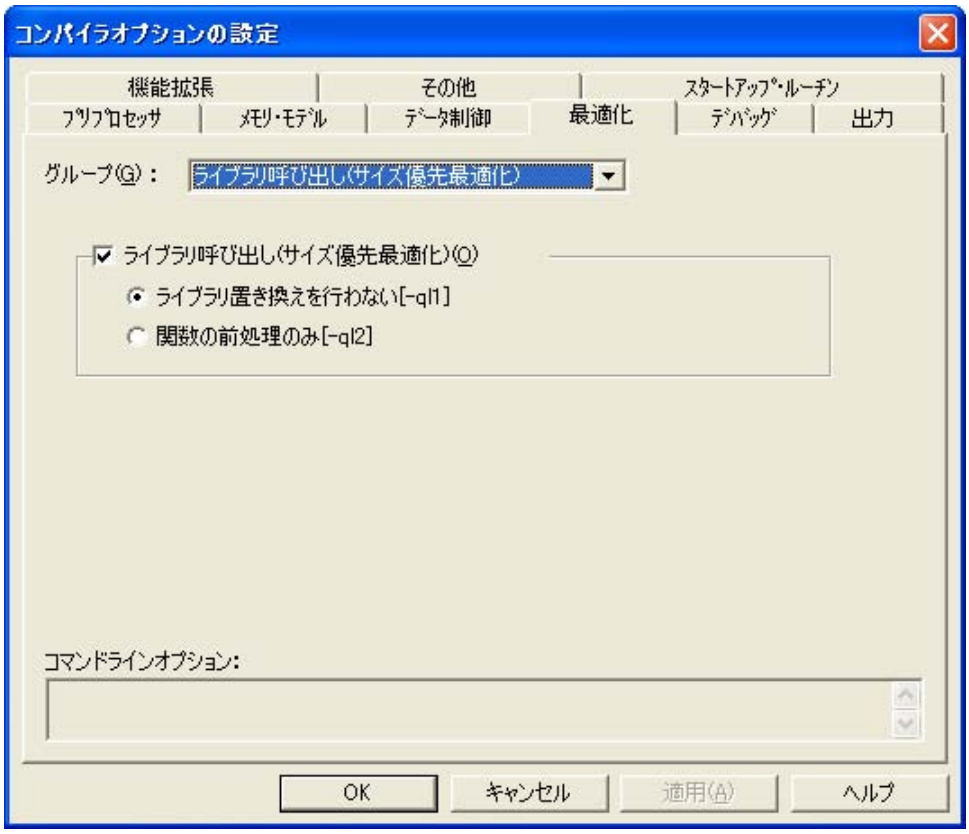
-qv オプションを有効にする場合に，チェック・ボックスをチェックします。

- 分岐命令の最適化を行う [-qj](J)

-qj オプションを有効にする場合に，チェック・ボックスをチェックします。

(3)“グループ (G)”で“ライブラリ呼び出し（サイズ優先最適化）”を選択した場合

図 3-12 [コンパイラオプションの設定] ダイアログ（“ライブラリ呼び出し（サイズ優先最適化）”選択時）

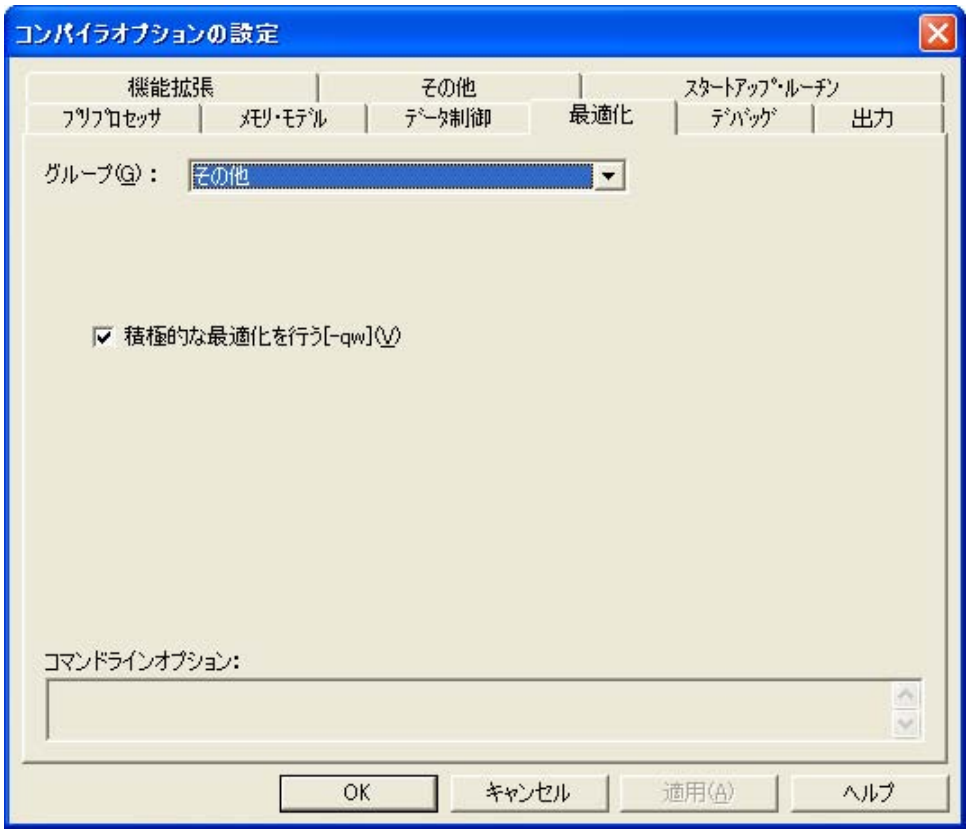


- ライブラリ呼び出し（サイズ優先最適化）(O)

-ql オプションを有効にする場合にチェック・ボックスをチェックし，オブジェクト・サイズ優先最適化の強度をラジオ・ボタンで指定します。-qln の n の数字が大きくなるにつれて，オブジェクト・コード・サイズが小さくなりますが，実行速度がそれに伴って遅くなります。

(4)“グループ (G)”で“その他”を選択した場合

図 3-13 [コンパイラオプションの設定] ダイアログ（“その他”選択時）

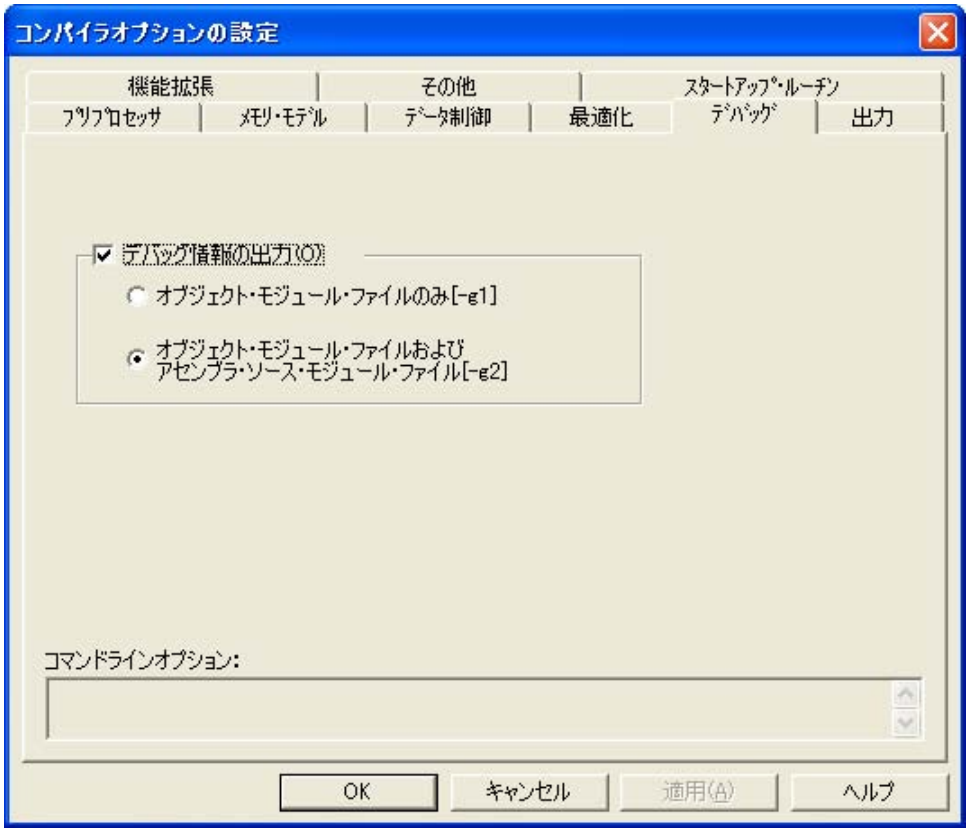


- 積極的な最適化を行う [-qw](V)

-qw オプションを有効にする場合に，チェック・ボックスをチェックします。

[デバッグ] タブ

図 3-14 [コンパイラオプションの設定] ダイアログ（[デバッグ] タブ選択時）



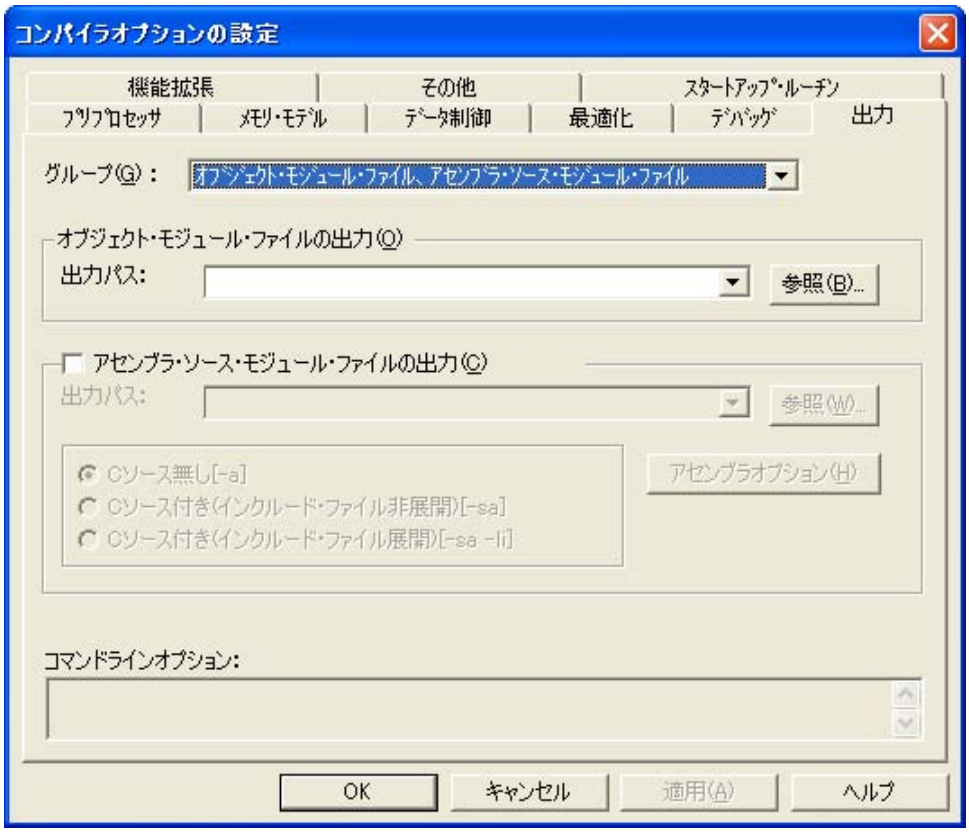
- デバッグ情報の出力 (O)

-g オプションを有効にする場合にチェック・ボックスをチェックし，デバッグ情報を出力するファイルをラジオ・ボタンで選択します。

[出力] タブ

(1) “グループ (G)” で “オブジェクト・モジュール・ファイル、アセンブラ・ソース・モジュール・ファイル” を選択した場合

図 3-15 [コンパイラオプションの設定] ダイアログ（“オブジェクト・モジュール・ファイル、アセンブラ・ソース・モジュール・ファイル” 選択時）



- オブジェクト・モジュール・ファイルの出力 (O)

オブジェクト・モジュール・ファイルの出力パスを指定する場合に，コンボ・ボックスにパス名を入力します。

コンボ・ボックスに入力可能な文字数は，259 文字です。

[参照 (B)...] ボタンから指定することもできます（[フォルダの参照] ダイアログをオープンします）。

全体オプション指定時は，常にパス名が指定されたものとして処理を行います。

ソース・ファイルが指定されている場合は，パスが存在すればパス名として処理し，存在しなければファイル名として処理をします。

- アセンブラ・ソース・モジュール・ファイルの出力 (C)

-a/-sa/-li オプションを有効にする場合にチェック・ボックスをチェックし，アセンブラ・ソース・モジュール・ファイルに付加する C ソースの有無，インクルード・ファイル内容の有無をラジオ・ボタンで選択します。

また，アセンブラ・ソース・モジュール・ファイルの出力パスを指定する場合には，コンボ・ボックスにパス名を入力します。ソース・ファイル名として指定する場合は，拡張子は “asm” にしてください。

コンボ・ボックスに入力可能な文字数は，259 文字です。

[参照 (B)...] ボタンから指定することもできます（[フォルダの参照] ダイアログをオープンします）。

全体オプション指定時は，常にパス名が指定されたものとして処理を行います。

ソース・ファイルが指定されている場合は，パスが存在すればパス名として処理し，存在しなければファイル名として処理をします。

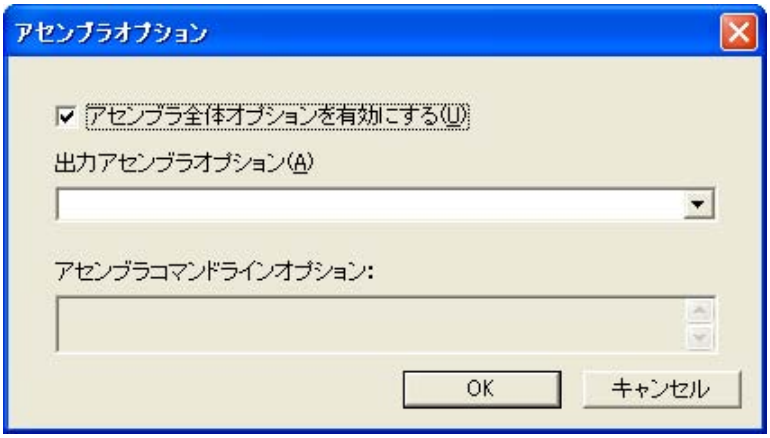
- [アセンブラオプション (H)] ボタン

アセンブラ・ソース・モジュール・ファイルに対し，アセンブラオプションを指定します。

指定しない場合は，アセンブラの全体オプションが指定されたものとして処理します。

なお，[アセンブラオプション (H)] ボタンを押すと，次のダイアログが表示されます。

図 3-16 [アセンブラオプション] ダイアログ



- アセンブラ全体オプションを有効にする (U)

[アセンブラオプションの設定] ダイアログで設定されている全体オプションを有効にする場合に，チェック・ボックスをチェックします。

- 出力アセンブラオプション (A)

C コンパイラの出力アセンブラ・ソース・ファイルに対してオプションを有効にする場合に，オプション名を含めた文字列をコンボ・ボックスに入力します。

コンボ・ボックスに入力可能な文字数は，259 文字です。

注意 デバイス種別指定（-c），デバイスファイル指定（-y），およびパラメータ・ファイル指定（-f）は，ツール DLL で設定するため，記述しないでください。

- アセンブラコマンドラインオプション

このエディット・ボックスは，読み取り専用です。

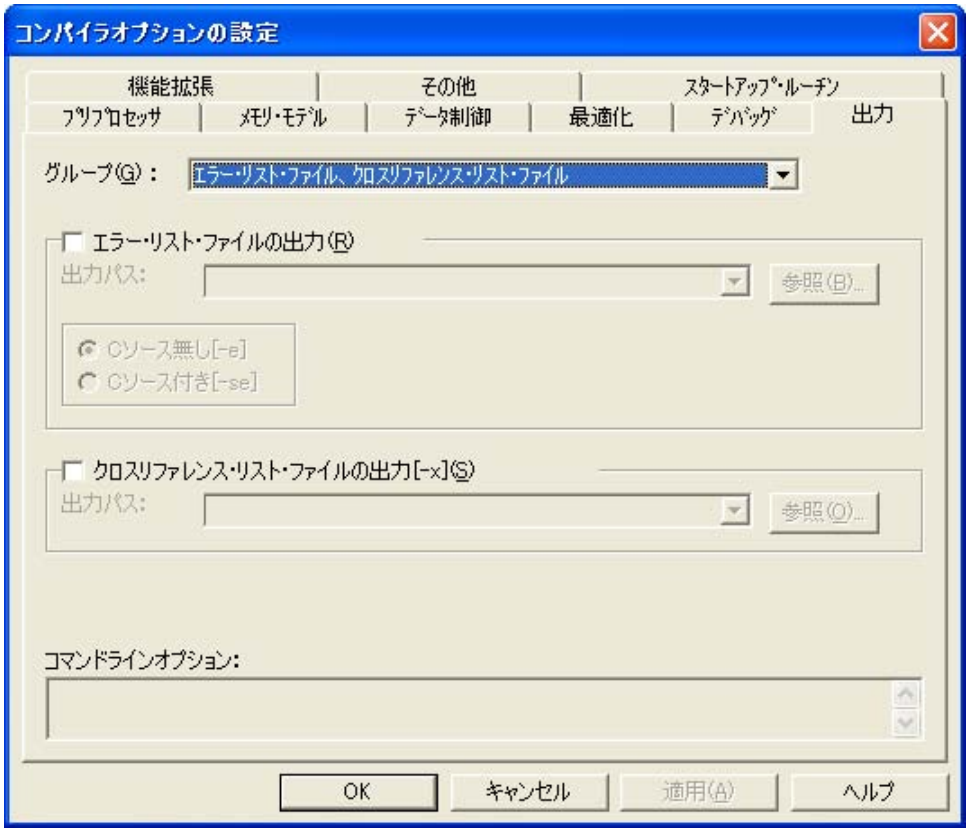
現在設定されているオプション文字列が表示されます。

対象オプションは，アセンブラ全体オプションと出力アセンブラオプションです。

各オプション設定ダイアログのラジオ・ボタン，チェック・ボックス，コンボ・ボックス等で指定されたオプション文字列は，直ちにこのエディット・ボックスに表示されます。

(2)“グループ (G)”で“エラー・リスト・ファイル、クロスリファレンス・リスト・ファイル”を選択した場合

図 3-17 [コンパイラオプションの設定] ダイアログ（“エラー・リスト・ファイル、クロスリファレンス・リスト・ファイル”選択時）



- エラー・リスト・ファイルの出力 (R)

-e/-se オプションを有効にする場合にチェック・ボックスをチェックし，エラー・リストに C ソースを付加する／付加しないをラジオ・ボタンで選択します。

また，エラー・リスト・ファイルの出力パスを指定する場合には，コンボ・ボックスにパス名を入力します。

コンボ・ボックスに入力可能な文字数は，259 文字です。

[参照 (B)...] ボタンから指定することもできます（[フォルダの参照] ダイアログをオープンします）。

全体オプション指定時は，常にパス名が指定されたものとして処理を行います。

ソース・ファイルが指定されている場合は，パスが存在すればパス名として処理し，存在しなければファイル名として処理をします。

- クロスリファレンス・リスト・ファイルの出力 [-x](S)

-x オプションを有効にする場合にチェック・ボックスをチェックします。

また，クロスリファレンス・リスト・ファイルの出力パスを指定する場合には，コンボ・ボックスにパス名を入力します。

コンボ・ボックスに入力可能な文字数は，259 文字です。

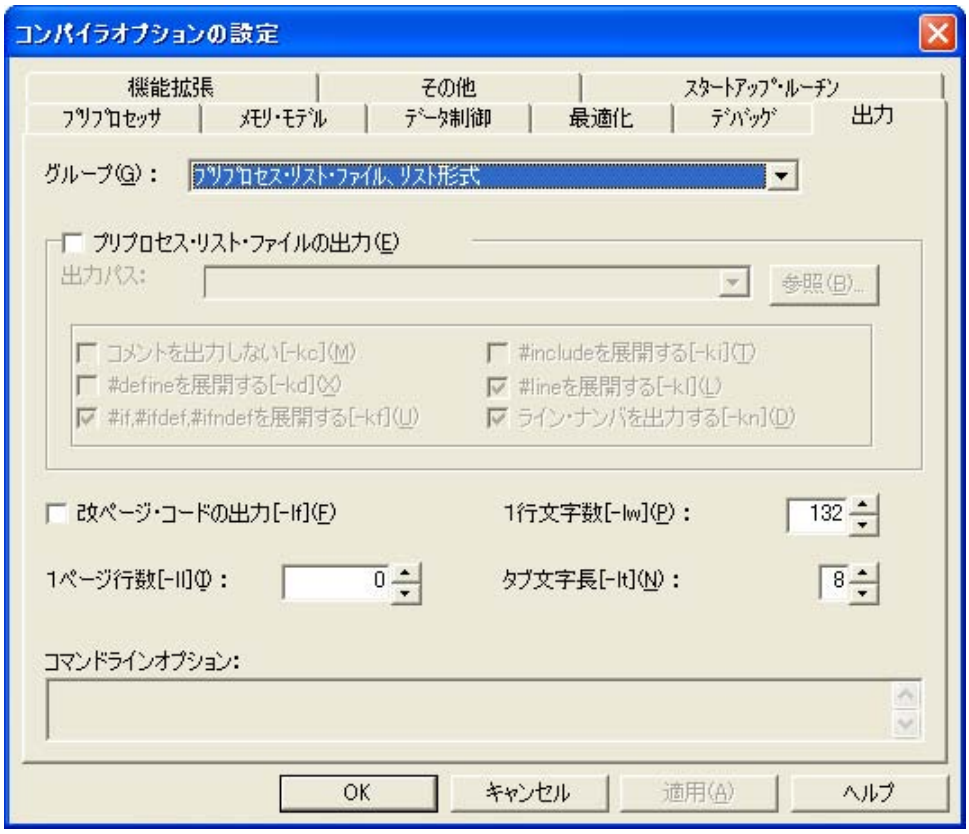
[参照 (O)...] ボタンから指定することもできます（[フォルダの参照] ダイアログをオープンします）。

全体オプション指定時は，常にパス名が指定されたものとして処理を行います。

ソース・ファイルが指定されている場合は，パスが存在すればパス名として処理し，存在しなければファイル名として処理をします。

(3)“グループ (G)”で“プリプロセス・リスト・ファイル、リスト形式”を選択した場合

図 3-18 [コンパイラオプションの設定] ダイアログ（“プリプロセス・リスト・ファイル、リスト形式”選択時）



- プリプロセス・リスト・ファイルの出力 (E)

-p オプション，および次のプリプロセス・リスト・ファイルに関する各指定を有効にする場合に，チェックします。

コメントを出力しない [-kc](M)

-kc オプションを有効にする場合に，チェック・ボックスをチェックします。

#define を展開する [-kd](X)

-kd オプションを有効にする場合に，チェック・ボックスをチェックします。

#if , #ifdef , #ifndef を展開する [-kf](U)

-kf オプションを有効にする場合に，チェック・ボックスをチェックします。

#include を展開する [-ki](T)

-ki オプションを有効にする場合に，チェック・ボックスをチェックします。

#line を展開する [-kl](L)

-kl オプションを有効にする場合に，チェック・ボックスをチェックします。

ライン・ナンバを出力する [-kn](D)

-kn オプションを有効にする場合に，チェック・ボックスをチェックします。

プリプロセス・リスト・ファイルの出力パスを指定する場合には，コンボ・ボックスにパス名を入力します。

コンボ・ボックスに入力可能な文字数は，259 文字です。

[参照 (B)...] ボタンから指定することもできます（[フォルダの参照] ダイアログをオープンします）。

全体オプション指定時は，常にパス名が指定されたものとして処理を行います。

ソース・ファイルが指定されている場合は，パスが存在すればパス名として処理し，存在しなければファイル名として処理をします。

- 改ページ・コードの出力 [-lf](F)

-lf オプションを有効にする場合に，チェック・ボックスをチェックします。

- 1 行文字数 [-lw](P)

-lw オプションで 1 行の文字数を指定します。

指定可能な文字数の範囲は，72 ～ 132 です。

- 1 ページ行数 [-ll](L)

-ll オプションで 1 ページの行数を指定します。

指定可能な行数の範囲は，0，および 20 ～ 32,767 です。

- タブ文字長 [-lt](N)

-lt オプションでタブの長さを指定します。

指定可能なタブの長さの範囲は，0 ～ 8 です。

[機能拡張] タブ

図 3-19 [コンパイラオプションの設定] ダイアログ（[機能拡張] タブ選択時）

- 言語仕様制御

ANSI 準拠 [-za](I)

-za オプションを有効にする場合に，チェック・ボックスをチェックします。

C++ コメントの使用を許可する [-zp](E)

-zp オプションを有効にする場合に，チェック・ボックスをチェックします。

コメントのネストを許可する [-zc](C)

-zc オプションを有効にする場合に，チェック・ボックスをチェックします。

関数の引数／返値を int 拡張しない [-zb](N)

-zb オプションを有効にする場合に，チェック・ボックスをチェックします。

コメント中の漢字コード (K)

ソースのコメント中で使用する漢字コードの種類（シフト JIS ／ EUC ／漢字コードなし）を，ラジオ・ボタンで選択します。

[その他] タブ

図 3-20 [コンパイラオプションの設定] ダイアログ（[その他] タブ選択時）

- 実行状態を表示する [-v](V)

-v オプションを有効にする場合に，チェック・ボックスをチェックします。

- ワーニング・レベル [-w](W)

-w オプションでワーニング・レベルを指定します。

指定可能なワーニング・レベルの範囲は，0 ～ 2 です。

レベル	説明
0	ワーニング・メッセージを出力しません。
1	通常のワーニング・メッセージを出力します。
2	詳細なワーニング・メッセージを出力します。

- コマンドファイルの作成 (U)

このチェック・ボックスを選択することにより，オプション文字列はコマンド・ファイルに出力されるため，オプション文字列の長さの制限を意識する必要がなくなります。

注意 ソース・ファイル単位で個別コンパイラ・オプションを設定する際は，本項目の設定を行うことはできません。

- テンポラリ・ファイル・パス [-t](M)

-t オプションで指定するテンポラリ・ファイルを格納するフォルダを，コンボ・ボックスに入力します。

指定可能なフォルダは，1 個です。

コンボ・ボックスに入力可能な文字数は，259 文字です。

[参照 (B)...] ボタンから指定することもできます（[フォルダの参照] ダイアログをオープンします）。

- パラメータ・ファイル (P)

-f オプションで指定するパラメータ・ファイル名を，コンボ・ボックスに入力します。

指定可能なフォルダは，1 個です。

コンボ・ボックスに入力可能な文字数は，259 文字です。

[参照 (O)...] ボタンから指定することもできます（[ParameterFile] ダイアログをオープンします）。

- その他のオプション (T)

各オプション指定項目以外のコンパイラ・オプションを指定する必要がある場合に，コンボ・ボックスにオプションを入力します。

コンボ・ボックスに入力可能な文字数は，259 文字です。

- [リセット (E)] ボタン

オプション設定をデフォルト状態に戻します。

- [オプション情報読込 (R)...] ボタン

オプション設定を保存したオプション情報ファイルを読み込みます。

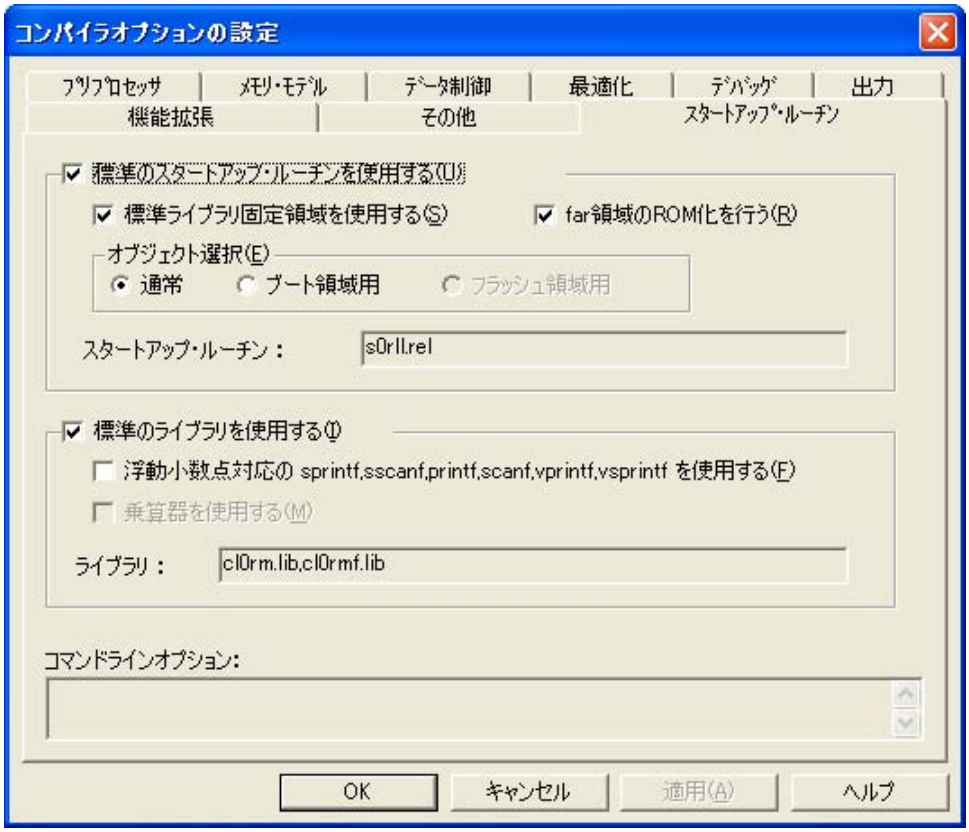
- [オプション情報保存 (S)...] ボタン

オプション設定を，オプション情報ファイルとして保存します。

なお，本ボタンは，[OK] ボタン，または [適用 (A)] ボタンで設定情報が決定されている場合のみ，有効となります。

[スタートアップ・ルーチン] タブ

図 3-21 [コンパイラオプションの設定] ダイアログ（[スタートアップ・ルーチン] タブ選択時）



注意 ソース・ファイル単位で個別コンパイラ・オプションを設定する際は，[スタートアップ・ルーチン] タブの設定を行うことはできません。

- 標準のスタートアップ・ルーチンを使用する (U)

C コンパイラが用意する標準のスタートアップ・ルーチンを使用する場合に，チェック・ボックスをチェックします。

標準ライブラリ固定領域を使用する (S)

標準ライブラリが使用する固定領域を使用する場合に，チェック・ボックスをチェックします。

far 領域の ROM 化を行う (R)

far 領域の ROM 化を行う場合に，チェック・ボックスをチェックします。

オブジェクト選択 (E)

通常／ブート領域用／フラッシュ領域用のスタートアップ・ルーチンを，ラジオ・ボタンで選択します。

[メモリ・モデル] タブの“フラッシュ用オブジェクトを出力する [-zf](U)”チェック・ボックスをチェックしていない場合は，通常／ブート領域用のスタートアップ・ルーチンを選択することができ，チェック・ボックスをチェックしている場合は，フラッシュ領域用のスタートアップ・ルーチンのみを選択することができます。

スタートアップ・ルーチン表示領域

使用するスタートアップ・ルーチンのファイル名を表示します。

- 標準のライブラリを使用する (I)

C コンパイラが用意する標準のライブラリを使用する場合に，チェック・ボックスをチェックします。

浮動小数点対応の sprintf, sscanf, printf, scanf, vprintf, vsprintf を使用する (F)

浮動小数点対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf 関数を使用する場合に，チェック・ボックスをチェックします。

乗算器を使用する (M)

乗算器を持つ品種で，乗算器を使用する場合，チェック・ボックスにチェックします。

注意　乗算器を持たない品種の場合，選択することはできません。

ライブラリ表示領域

使用するライブラリのファイル名を表示します。

3.2 手順（セルフ書き換えモード未使用時）

3.2.1 PM+ からのビルド

PM+ を使用してビルドする方法を説明します。

PM+ は，開発環境の中心として，ツール群を統合管理するソフトウエアです。PM+ を使用することで，アプリケーション・プログラムや環境設定をプロジェクトとして扱い，エディタでのプログラム作成，ソース管理，コンパイル，デバッグを連続した作業として行うことができるようになります。

(1) PM+ の起動

各開発ツール・パッケージを正常にインストールすると，[スタート] ボタンのプログラム・フォルダに [NEC Electronics Tools] メニューが作成され，その中に PM+ などが登録されています。

PM+ を起動するには，メニューから [PM+] をクリックします。

(2) プロジェクトの作成

PM+ を使用して一連の開発作業を進めるには，プロジェクトを登録することから始めます。

プロジェクトを登録するためには，まず，それを管理する“ワークスペース”を作成する必要があります。

ワークスペースの作成方法については，PM+ のユーザーズ・マニュアルを参照してください。

(3) C コンパイラ，リンカのオプション設定

プロジェクト作成の終了時に自動的に作成されるメイク・ファイルは，ビルドのために必要最低限のオプションのみが設定されています。プロジェクト特有のオプションは，[ツール (T)] メニューで設定します。

[ツール (T)] メニューの [コンパイラオプションの設定 (C)] を選択すると，[コンパイラオプションの設定] ダイアログが表示されます。

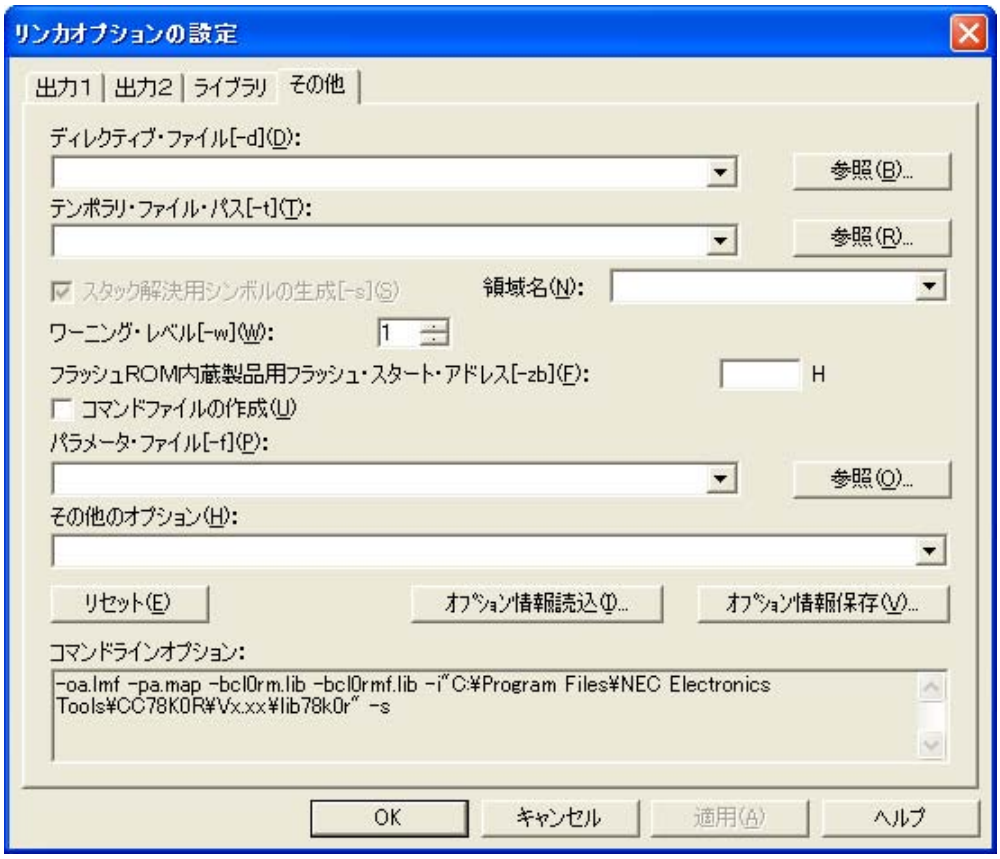
ここでは，最適化オプションをデフォルト [-qcjlvw] から，スピード優先 [-qx1] に変更した例を示します。

[コンパイラオプションの設定] ダイアログの [スタートアップ・ルーチン] タブで，標準のスタートアップ・ルーチンを使用する設定にしている場合は，C コンパイラ標準のスタートアップ・ルーチンが，すべてのソースの前でリンクされます（[リンカオプションの設定] ダイアログに表示はされません）。

標準のライブラリを使用する設定にしている場合には，C コンパイラ標準のライブラリがすべてのライブラリの後ろにリンクされます。

ソース・ファイルの設定で，C ソースが含まれる場合は，スタック・シンボル自動生成オプション -s が，リンカに自動的に指定されます。

スタートアップ・ルーチンのファイル名は，ロード・モジュール・ファイル名に影響を与えません。



(4) プロジェクトのビルド

設定したオプションで，プロジェクトをビルドします。

プロジェクト全体のビルドは，[ビルド (B)] →[ビルド (B)] メニュー，またはツールバーの [ビルド] ボタンを押すことにより行います。自動的に作成されるメイク・ファイルによって，PM+ のメイクが起動されます。

ビルドが終了すると，メッセージ・ダイアログが表示されます。正常終了したことを確認してください。

注意 ビルド時に [OutPut] ウインドウに表示された内容は，プロジェクト・フォルダに“プロジェクト・ファイル名 +.plg”というファイル名で保存されます。

3.2.2 コマンド行（コマンド・プロンプト）でのコンパイル～リンクの実行手順

(1) パラメータ・ファイル未使用時

コマンド行で C コンパイラ，アセンブラ，リンカを起動する際には，次のコマンドを使用します。
なお，C ソース中にアセンブラ記述がない場合は，アセンブルは必要ありません。C コンパイラが出力したオブジェクト・モジュール・ファイルをリンクしてください（△：空白）。

```
>[ パス名 ]cc78k0r[ △オプション ] △ C ソース名 [ △オプション ]
>[ パス名 ]ra78k0r[ △オプション ] △アセンブラ・ソース名 [ △オプション ]
>[ パス名 ]lk78k0r[ △オプション ] △オブジェクト・モジュール名 [ △オプション ]
```

注意 ユーザが作成したライブラリをリンクする場合は，CC78K0R 付属のライブラリと浮動小数点用ライブラリを，必ずライブラリ並びの最後に指定してください。
浮動小数点対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf 関数を使用する場合は，CC78K0R 付属の浮動小数点用ライブラリ，CC78K0R 付属のライブラリの順で指定してください。
浮動小数点未対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf 関数を使用する場合は，CC78K0R 付属のライブラリ，CC78K0R 付属の浮動小数点用ライブラリの順で指定してください。
また，CC78K0R 付属のスタートアップ・ルーチンをユーザ・プログラムの前に指定するようにしてください。

次に，リンク時のライブラリ，オブジェクト・モジュール・ファイルの指定順序を示します。

（ライブラリ指定順序）

- 浮動小数点未対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf 関数を使用する場合
  (a) ユーザ・プログラムのライブラリ・ファイル（-b オプションで指定）
  (b) CC78K0R 付属のライブラリ・ファイル（-b オプションで指定）
  (c) CC78K0R 付属の浮動小数点用ライブラリ・ファイル（-b オプションで指定）
- 浮動小数点対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf 関数を使用する場合
  (a) ユーザ・プログラムのライブラリ・ファイル（-b オプションで指定）
  (b) CC78K0R 付属の浮動小数点用ライブラリ・ファイル（-b オプションで指定）
  (c) CC78K0R 付属のライブラリ・ファイル（-b オプションで指定）

（その他のファイル指定順序）

(a) CC78K0R 付属のスタートアップ・ルーチンのオブジェクト・ファイル
(b) ユーザ・プログラムのオブジェクト・モジュール・ファイル

次に，C ソース s1.c とアセンブラ・ソース s2.asm をリンクする例を示します。

```
C>cc78k0r -cf1166a0 s1.c -e -a
-i"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥inc78k0r"
-y"C:¥Program Files¥NEC Electronics Tools¥dev"
C>ra78k0r -cf1166a0 s2.asm -e
-y"C:¥Program Files¥NEC Electronics Tools¥dev"
C>lk78k0r s0rll.rel s01.rel s2.rel
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rxm.lib"
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rm.lib" -s
-osample.lmf -y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

備考 複数のコンパイラ・オプションを指定する場合には，それぞれのコンパイラ・オプションの間を空白で区切ります。オプションの記述は，英大文字，英小文字のいずれでもかまいません。詳細については，「第 5 章 コンパイラ・オプション」を参照してください。

-i オプション指定，-b オプションのパス指定，および -y オプション指定は，条件によっては省略することができます。詳細については，「第 5 章 コンパイラ・オプション」，および「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアルを参照してください。

(2) パラメータ・ファイル使用時

C コンパイラ，アセンブラ，リンカ起動時に複数のオプションを入力する際に，コマンド行で起動に必要な情報を指定しきれない場合，また同じ指定を何回も繰り返す場合があります。このようなときに，パラメータ・ファイルを使用します。
パラメータ・ファイルを使用する場合は，パラメータ・ファイル指定オプション -f をコマンド行の中で指定してください。

パラメータ・ファイルによる起動方法は，次のようになります。

```
>[ パス名 ]cc78k0r △ -f パラメータ・ファイル名
>[ パス名 ]ra78k0r △ -f パラメータ・ファイル名
>[ パス名 ]lk78k0r △ -f パラメータ・ファイル名
```

次に使用例を示します。

```
C>cc78k0r -fpara.pcc
C>ra78k0r -fpara.pra
C>lk78k0r -fpara.plk
```

パラメータ・ファイルは，エディタで作成します。コマンド行で指定すべきすべてのオプション，出力ファイル名を記述することができます。
パラメータ・ファイルをエディタで作成した例を次に示します。

＜ para.pcc の内容＞

```
-cf1166a0 s1.c -e -a
-i"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥inc78k0r"
-y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

＜ para.pra の内容＞

```
-cf1166a0 s2.asm -e -y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

＜ para.plk の内容＞

```
s0rll.rel s1.rel s2.rel
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rxm.lib"
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rm.lib" -s
-osample.lmf -y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

-i オプション指定，-b オプションのパス指定，および -y オプション指定は，条件によっては省略することができます。詳細については，「第 5 章 コンパイラ・オプション」，および「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアルを参照してください。

3.3 手順（セルフ書き換えモード使用時）

この機能は，フラッシュ・メモリのセルフ書き換え機能を持つデバイスにのみ，使用することができます。

3.3.1 PM+ からのコンパイルからリンク

PM+ を使用してメイクする方法を示します。

必ず次のような順番で，コンパイルからリンクを行ってください。

(1) ブート領域用プログラムのコンパイル～リンク

(a) プロジェクトの作成

ブート領域用プロジェクトを作成し，ソース・ファイルを登録してください。

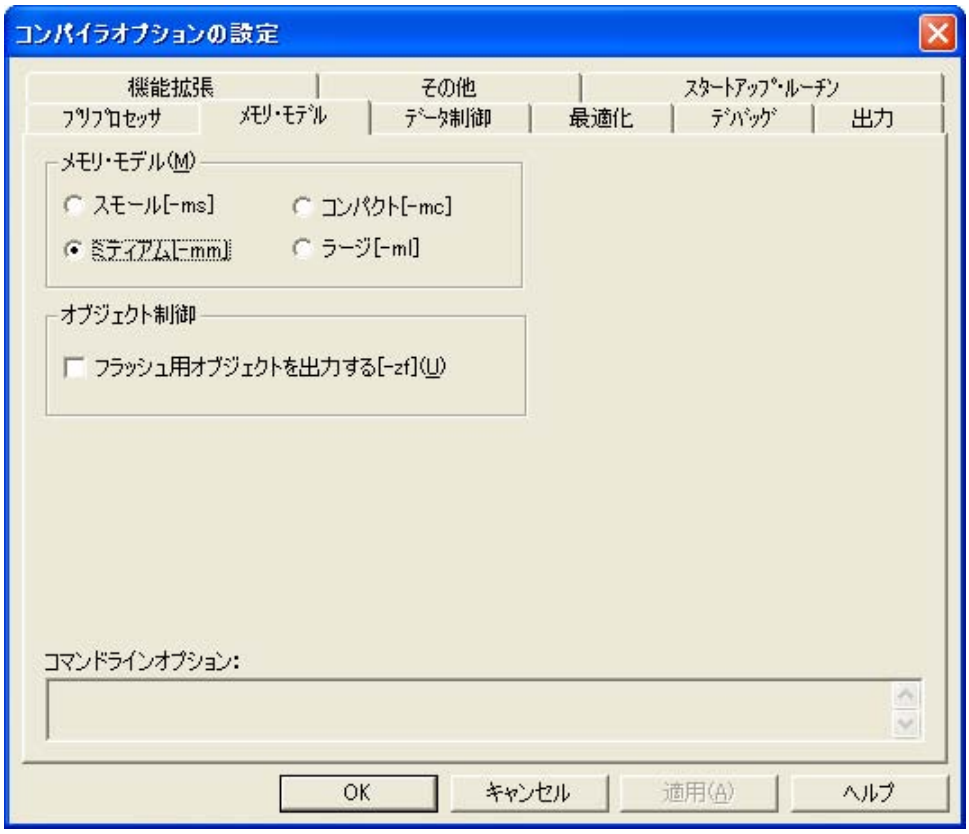
(b) C コンパイラ，リンカ，オブジェクト・コンバータのオプション設定

プロジェクト作成の終了時に自動的に作成されるメイク・ファイルは，ビルドのために必要最低限のオプションのみが設定されています。プロジェクト特有のオプションは，[ツール (T)] メニューで設定します。

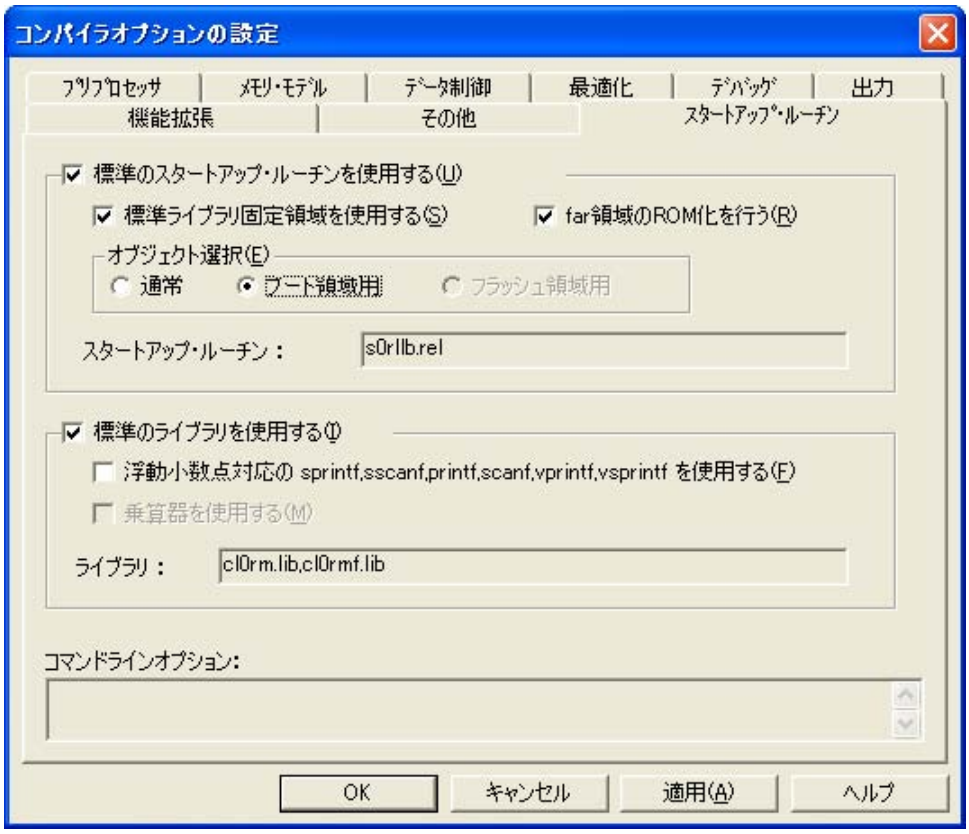
[ツール (T)] メニューの [コンパイラオプションの設定 (C)] を選択すると，[コンパイラオプションの設定] ダイアログが表示されます。

(i) C コンパイラ・オプションの設定

[メモリ・モデル] タブで，“フラッシュ用オブジェクトを出力する [-zf](U)”チェック・ボックスはチェックしないでください。



[スタートアップ・ルーチン] タブの“オブジェクト選択 (E)”で，“ブート領域用”ラジオ・ボタンを選択してください。



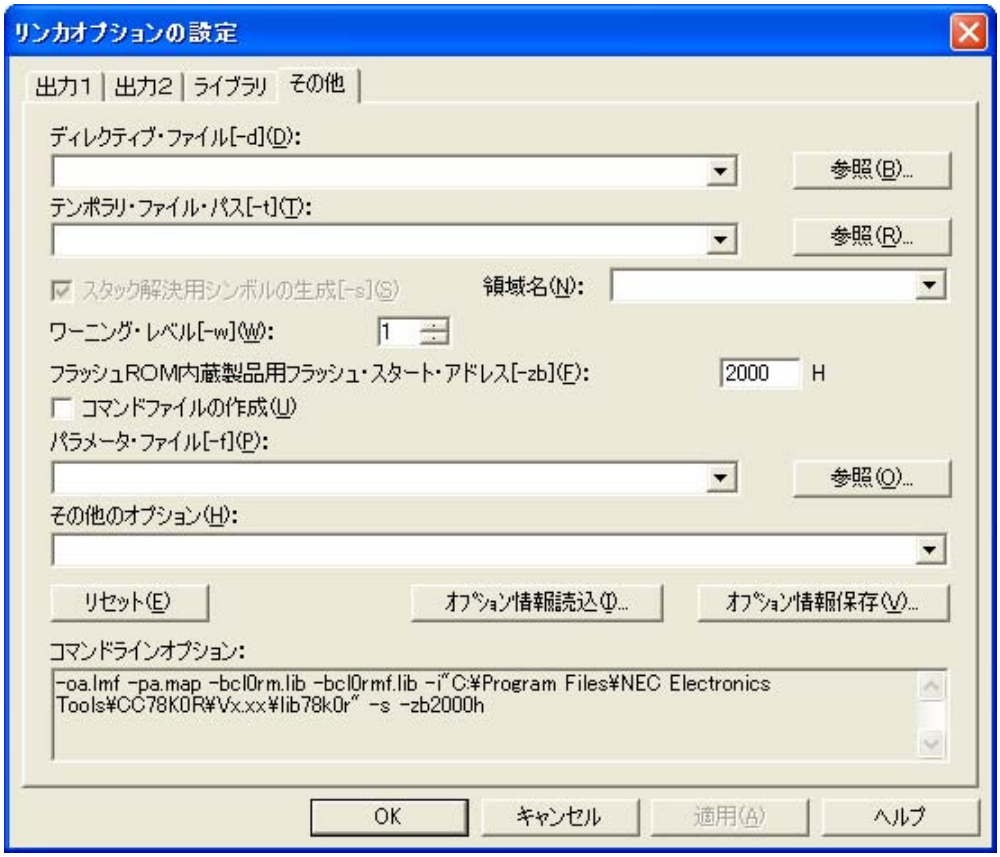
(ii) リンカ・オプションの設定

“フラッシュ ROM 内蔵製品用フラッシュ・スタート・アドレス [-zb](E)”を指定し，[OK] ボタンを押します。

コンパイラ・オプション設定の [スタートアップ・ルーチン] タブで，標準のスタートアップ・ルーチンと標準のライブラリを使用する設定にしているので，リンカ・オプションの設定で，スタートアップ・ルーチン，ライブラリを指定する必要はありません。

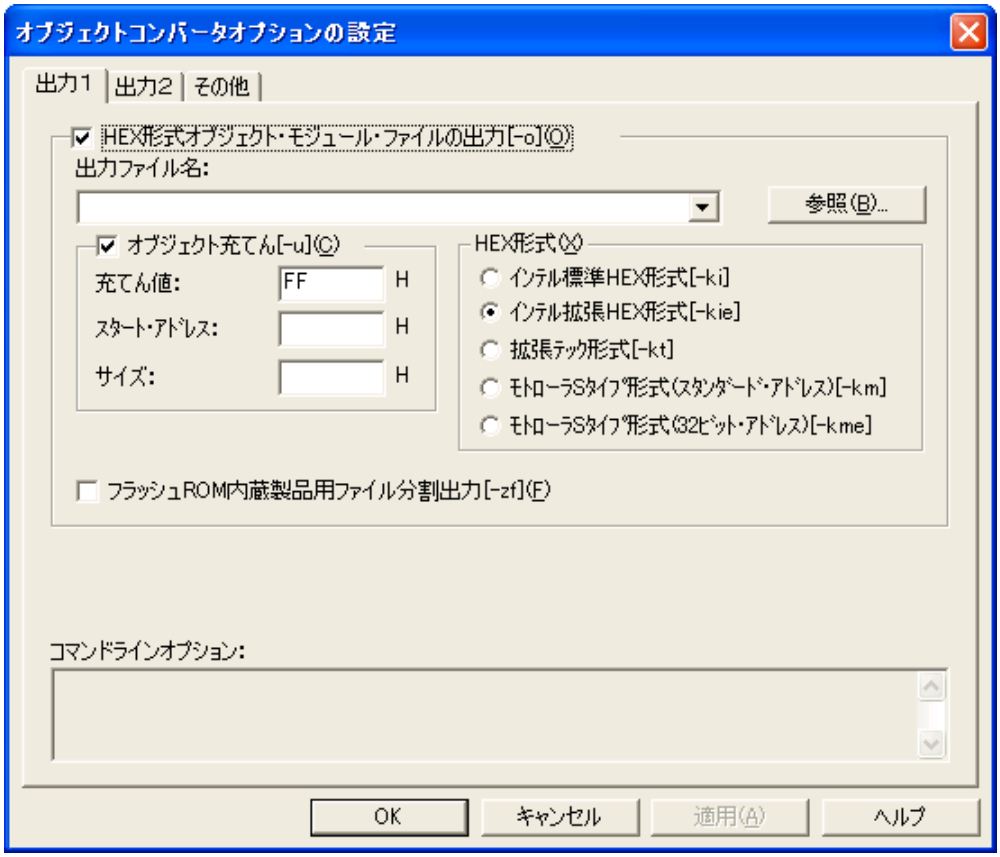
また，ソースファイル指定に，C ソース（boot.c）が含まれているので，“スタック解決用シンボルの生成 [-s](S)”オプションが自動的に設定されます。

備考 リンカ・オプションについては，「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアルを参照してください。



(iii) オブジェクト・コンバータ・オプションの設定

“フラッシュ ROM 内蔵製品用ファイル分割出力 [-zf](F)” チェック・ボックスをチェックしないでください。



注意 ブート領域用プログラムをコンパイルし，オブジェクト・コンバートしたあと，フラッシュ・ライタで HEX ファイル（例：boot.hex）を書き込みます。書き込み後は，上記の手順で作成したロード・モジュール・ファイル（例：boot.lmf）と HEX ファイルは必ず保存しておき，再度ブート領域用プログラムのビルドを行わないようにしてください。

(c) プロジェクトのビルド

設定したオプションで，プロジェクトをビルドします。

プロジェクト全体のビルドは，[ビルド (B)] メニュー→ [ビルド (B)] から，またはツール・バーの [ビルド] ボタンを押すことにより行います。自動的に作成したメイク・ファイルによって，PM+ のメイクが起動されます。

ビルドが終了すると，メッセージ・ダイアログが表示されます。正常終了したことを確認してください。

注意 ビルド時に [OutPut] ウインドウに表示された内容は，プロジェクト・ディレクトリに“プロジェクト・ファイル名 +.plg”というファイル名で保存されます。

(2) フラッシュ領域用プログラムのコンパイル～リンク

(a) プロジェクトの作成

フラッシュ領域用プロジェクトを作成し，ソース・ファイルを登録してください。

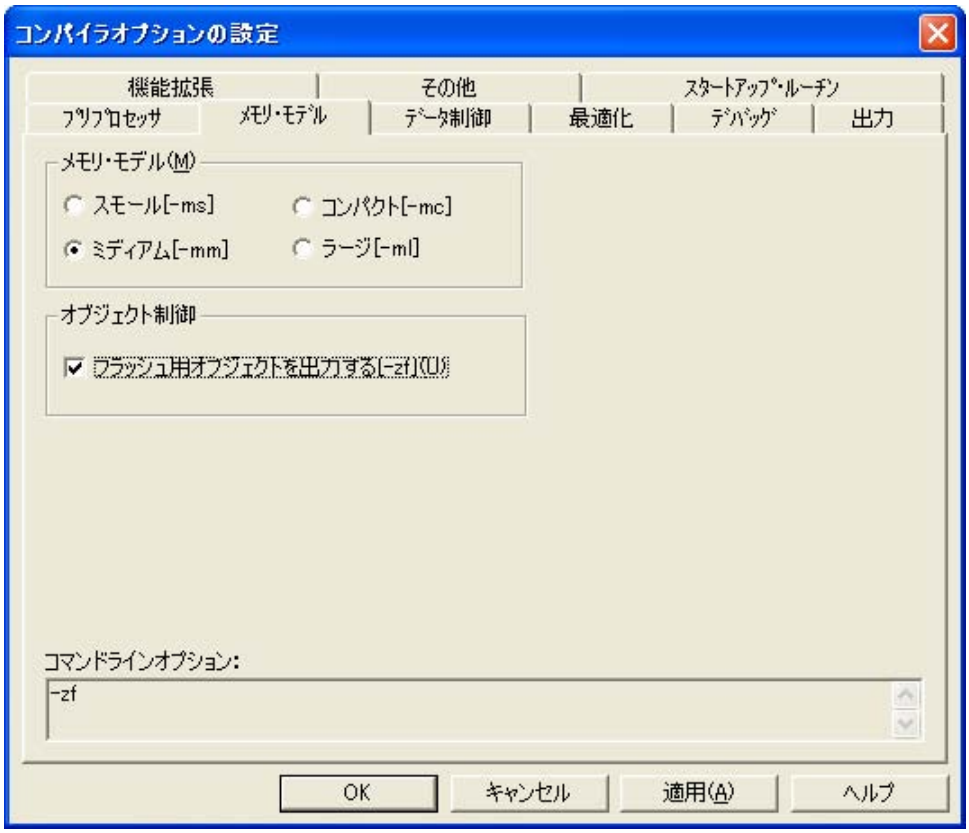
(b) C コンパイラ，リンカ，オブジェクト・コンバータのオプション設定

プロジェクト作成の終了時に自動的に作成されるメイク・ファイルは，ビルドのために必要最低限のオプションのみが設定されています。プロジェクト特有のオプションは，[ツール (T)] メニューで設定します。

[ツール (T)] メニューの [コンパイラ・オプションの設定 (C)] を選択すると，[コンパイラオプションの設定] ダイアログが現れます。

(i) コンパイラ・オプションの設定

“フラッシュ用オブジェクトを出力する [-zf](U)” チェック・ボックスをチェックしてください。



[スタートアップ・ルーチン] タブの “オブジェクト選択 (E)” で，“フラッシュ領域用” ラジオ・ボタンが自動選択されます。

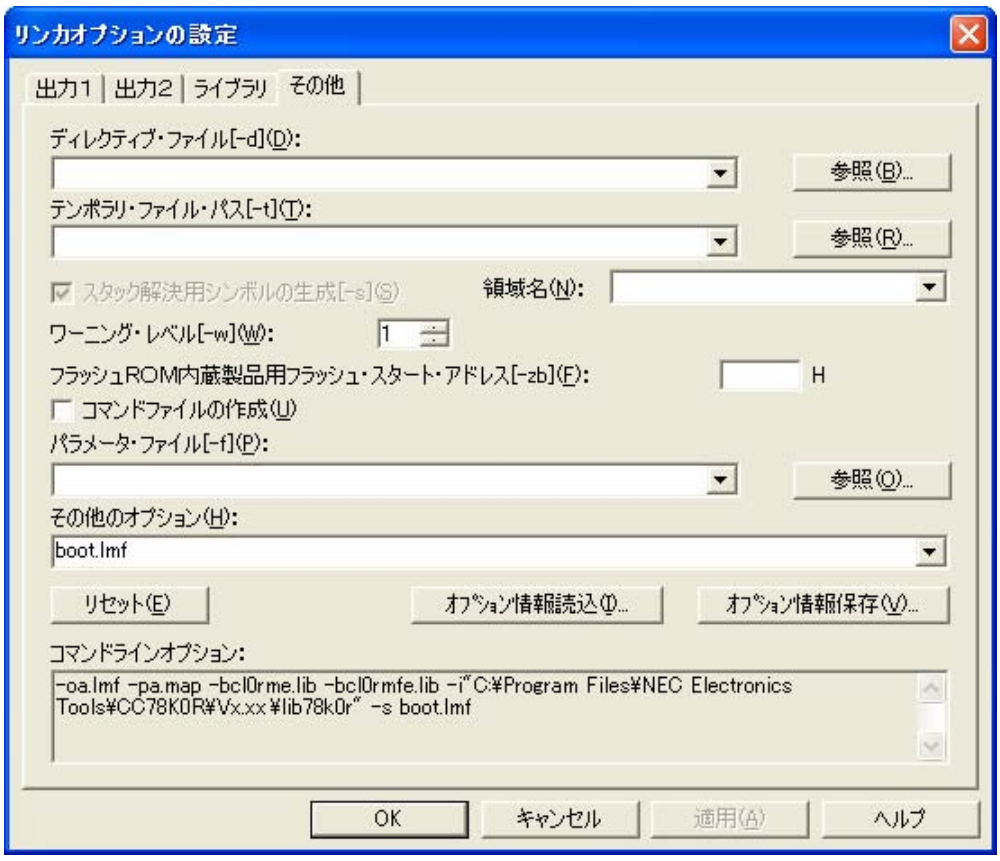
(ii) リンカ・オプションの設定

“その他のオプション (O)”コンボ・ボックスに，作成したブート領域用ロード・モジュール・ファイルを指定し，[OK] ボタンを押します。

コンパイラ・オプションの設定の [スタートアップ・ルーチン] タブで，標準のスタートアップ・ルーチンと標準のライブラリを使用する設定にしているので，リンカ・オプションの設定で，スタートアップ・ルーチンとライブラリを指定する必要はありません。

また，ソースファイル指定に C ソース（flash.c）が含まれているので，“スタック解決用シンボルの生成 [-s](S)”オプションが自動的に設定されます。

備考 リンカ・オプションについては，「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアルを参照してください。

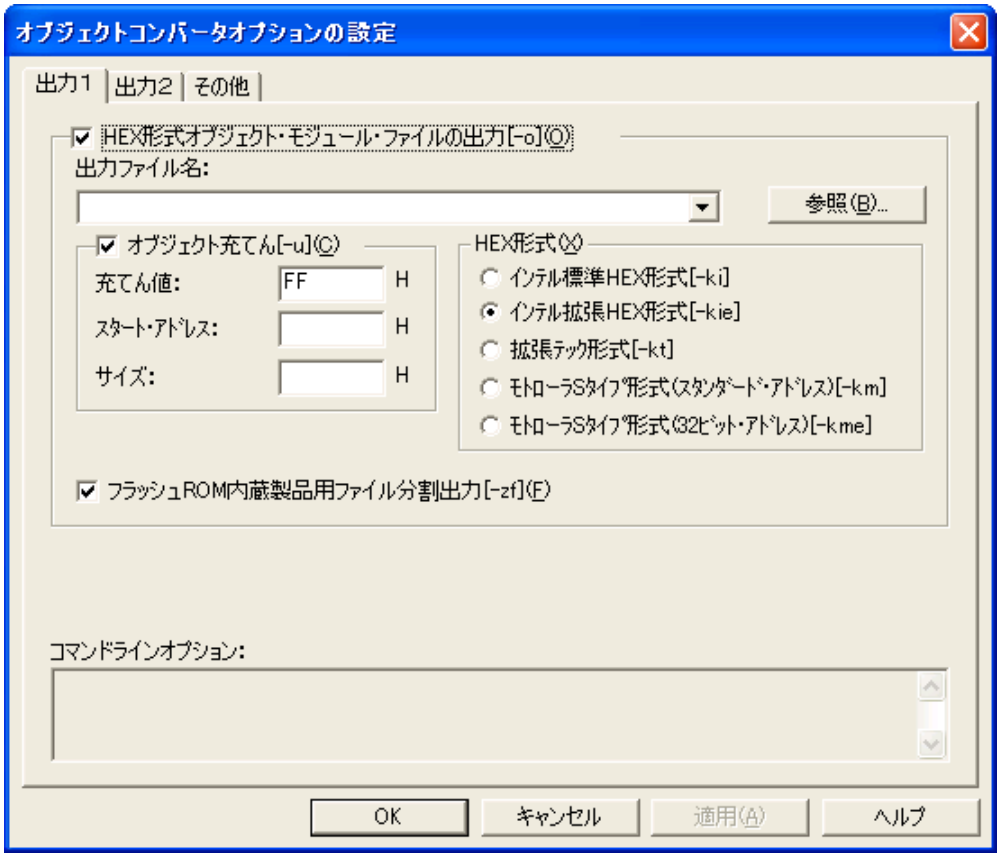


(iii) オブジェクト・コンバータ・オプションの設定（フラッシュ領域用）

“フラッシュ ROM 内蔵製品用ファイル分割出力 [-zf](F)” チェック・ボックスをチェックしてください。

ここで，-zf オプションを指定することにより，ブート領域用 HEX ファイル（例：flash.hxb）とフラッシュ領域用 HEX ファイル（例：flash.hxf）が出力されます。

flash.hxb とブート領域用プログラムのビルド時に生成された boot.hex は，同じ内容となりますが，ブート領域用 HEX ファイルがすでに書き込まれていて，フラッシュ領域用プログラムを再度ビルドした場合は，保存しておいた boot.hex と作成した flash.hxb で違いがないかを確認することをお勧めします。



(c) プロジェクトのビルド

設定したオプションで，プロジェクトをビルドします。

プロジェクト全体のビルドは，[ビルド (B)] メニュー→ [ビルド (B)] から，またはツール・バーの [ビルド] ボタンを押すことにより行います。自動的に作成したメイク・ファイルによって，PM+ のメイクが起動されます。

ビルドが終了すると，メッセージ・ダイアログが表示されます。正常終了したことを確認してください。

注意 ビルド時に [OutPut] ウインドウに表示された内容は，プロジェクト・ディレクトリに“プロジェクト・ファイル名 +.plg”というファイル名で保存されます。

3.3.2 コマンド行（コマンド・プロンプト）でのコンパイルからリンク

(1) パラメータ・ファイル未使用時

コマンド行で C コンパイラ，アセンブラ，リンカを起動する際には，次のコマンドを使用します。

なお，C ソース中にアセンブラ記述がない場合は，アセンブルは必要ありません。C コンパイラが出力したオブジェクト・モジュール・ファイルをリンクしてください（△：空白）。

```
>[ パス名 ]cc78k0r[ △オプション ] △ C ソース名 [ △オプション ]
>[ パス名 ]ra78k0r[ △オプション ] △アセンブラ・ソース名 [ △オプション ]
>[ パス名 ]lk78k0r[ △オプション ]] △オブジェクト・モジュール名など [ △オプション ]
```

次に，ブート領域用 C ソースとフラッシュ領域用 C ソースをコンパイル，リンクする例を示します。

(a) ブート領域用プログラムのコンパイルからリンク，オブジェクト・コンバート

＜例 1：ブート領域用プログラムのコンパイル＞

```
C>cc78k0r -cf1166a0 boot.c
-i"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥inc78k0r"
-y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

＜例 2：ブート領域用プログラムのリンク＞

```
C>lk78k0r s0rllb.rel boot.rel
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rxm.lib"
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rm.lib"
-s -oboot.lmf -zb2000h -y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

＜例 3：ブート領域用プログラムのオブジェクト・コンバート＞

```
C>oc78k0r boot.lmf -oboot.lmf -y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

注意 ブート領域用プログラムをコンパイルしオブジェクト・コンバートしたあと，フラッシュ・ライタで HEX ファイル（例：boot.hex）を書き込みます。書き込み後は，上記の手順で生成したロード・モジュール・ファイル（例：boot.lmf）と HEX ファイルは必ず保存しておき，再度ブート領域用プログラムのビルドを行わないようにしてください。

(b) フラッシュ領域用プログラムのコンパイルからリンク

＜例 1：フラッシュ領域用プログラムのコンパイル＞

```
C>cc78k0r -cf1166a0 flash.c -zf
-i"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥inc78k0r"
-y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

＜例 2：フラッシュ領域用プログラムのリンク＞

```
C>lk78k0r boot.lmf s0lle.rel flash.rel
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rxm.lib"
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rm.lib"
-s -oflash.lmf -y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

＜例 3：フラッシュ領域用プログラムのオブジェクト・コンバート＞

```
C>oc78k0r flash.lmf -oflash.lmf -y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

注意 オブジェクト・コンバート時に，-zf オプションを指定することにより，ブート領域用 HEX ファイル（例：flash.hxb）とフラッシュ領域用 HEX ファイル（例：flash.hxf）が出力されます。flash.hxb とブート領域用プログラムのビルド時に生成された boot.hex は，同じ内容となりますが，ブート領域用 HEX ファイルがすでに書き込まれていて，フラッシュ領域用プログラムを再度ビルドした場合は，保存しておいた boot.hex と作成した flash.hxb で違いがないかを確認することをお勧めします。

備考 複数のコンパイラ・オプションを指定する場合には，それぞれのコンパイラ・オプションの間を空白で区切ります。オプションの記述は，英大文字，英小文字のいずれでもかまいません。詳細については，「第 5 章 コンパイラ・オプション」を参照してください。

-i オプション指定，-b オプションのパス指定，および -y オプション指定は，条件によっては省略することができます。詳細については，「第 5 章 コンパイラ・オプション」，および「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアルを参照してください。

注意 ユーザが作成したライブラリ，浮動小数点用ライブラリをリンクする場合には，CC78K0R 付属のライブラリを必ずライブラリの並びの最後に指定してください。また，フラッシュ領域用プログラムとブート領域用プログラムをリンクする際には，ブート領域用ロード・モジュール・ファイルを最初に指定し，フラッシュ領域用スタートアップ・ルーチンをユーザ・プログラムの前に指定するようにしてください。

次にリンク時のライブラリ，オブジェクト・モジュール・ファイルの指定順序を示します。

（ライブラリ指定順序）

- 浮動小数点未対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf 関数を使用する場合

  (i) ユーザ・プログラムのライブラリ・ファイル（-b オプションで指定）

  (ii) CC78K0R 付属のライブラリ・ファイル（-b オプションで指定）

  (iii) CC78K0R 付属の浮動小数点用ライブラリ・ファイル（-b オプションで指定）

- 浮動小数点対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf 関数を使用する場合

  (i) ユーザ・プログラムのライブラリ・ファイル（-b オプションで指定）

  (ii) CC78K0R 付属の浮動小数点用ライブラリ・ファイル（-b オプションで指定）

  (iii) CC78K0R 付属のライブラリ・ファイル（-b オプションで指定）

注意 ブート領域用プログラムのリンク時には，ブート領域用ライブラリ，フラッシュ領域用プログラムのリンク時には，フラッシュ領域用ライブラリを指定してください。

（その他のファイル指定順序）

(i) ユーザ・プログラムのブート領域用ロード・モジュール・ファイル

(ii) CC78K0R 付属のフラッシュ領域用スタートアップ・ルーチンのオブジェクト・モジュール・ファイル

(iii) ユーザ・プログラムのフラッシュ領域用オブジェクト・モジュール・ファイル

(2) パラメータ・ファイル使用時

C コンパイラ，アセンブラ，リンカ起動時に複数のオプションを入力する際に，コマンド行で起動に必要な情報を指定しきれない場合，また同じ指定を何回も繰り返す場合があります。このようなときに，パラメータ・ファイルを使用します。

パラメータ・ファイルを使用する場合は，パラメータ・ファイル指定オプション -f をコマンド行の中で指定してください。

パラメータ・ファイルによる起動方法は，次のようになります。

```
>[ パス名 ]cc78k0r △ -f パラメータ・ファイル名
>[ パス名 ]ra78k0r △ -f パラメータ・ファイル名
>[ パス名 ]lk78k0r △ -f パラメータ・ファイル名
```

次に使用例を示します。

```
C>cc78k0r -fpara.pcc
C>lk78k0r -fpara.plk
```

パラメータ・ファイルは，エディタで作成します。コマンド行で指定すべきすべてのオプション，出力ファイル名を記述することができます。

パラメータ・ファイルをエディタで作成した例を次に示します。

＜ para.pcc の内容＞

```
-cf1166a0 boot.c
-i"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥inc78k0r"
-y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

＜ para.plk の内容＞

```
s0rllb.rel boot.rel
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rxm.lib"
-b"C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥lib78k0r¥cl0rm.lib"
-s -oboot.lmf -zb2000h
-y"C:¥Program Files¥NEC Electronics Tools¥dev"
```

備考 -i オプション指定，-b オプションのパス指定，および -y オプション指定は，条件によっては省略することができます。詳細については，「第 5 章 コンパイラ・オプション」，および「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアルを参照してください。

3.4 C コンパイラの入出力ファイル

CC78K0R は，C 言語で記述された C ソース・モジュール・ファイルを入力します。そして，それらを機械語に変換して，オブジェクト・モジュール・ファイルとして出力します。

また，コンパイル後にユーザがアセンブリ言語レベルで内容を確認，修正できるように，アセンブラ・ソース・モジュール・ファイルを出力します。さらに，コンパイラ・オプションにより，プリプロセス・リスト，クロスリファレンス・リスト，エラー・リストなどのリスト・ファイルを出力します。

コンパイル・エラーがある場合，エラー・メッセージをコンソール，エラー・リスト・ファイルなどに出力します。エラーがある場合は，エラー・リスト・ファイル以外の各種ファイルは出力されません。

CC78K0R の入出力ファイルを次に示します。

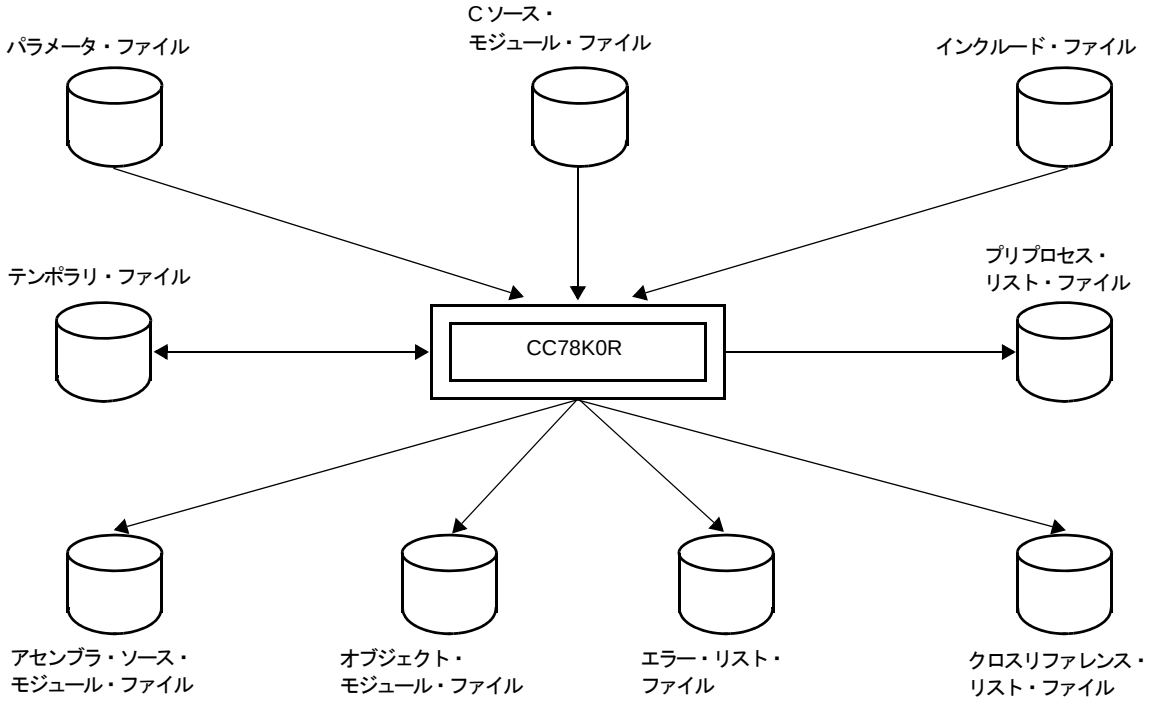
表 3-1 C コンパイラの入出力ファイル

種類	ファイル名	説明	デフォルト・ファイル・タイプ
入力ファイル	C ソース・モジュール・ファイル	- C 言語で記述されたソース・ファイル（ユーザ作成ファイル）	c
	インクルード・ファイル	- C ソース・モジュール・ファイルで参照するファイル - C 言語で記述されたファイル（ユーザ作成ファイル）	h
	パラメータ・ファイル	- C コンパイラ実行の際にコマンド行で指定不可能な多数のコマンドを指定したいときにユーザがエディタで作成するファイル	pcc
出力ファイル	オブジェクト・モジュール・ファイル	- 機械語情報と機械語の配置アドレスに関する再配置情報，およびシンボル情報を含んだバイナリ・イメージ・ファイル	rel
	アセンブラ・ソース・モジュール・ファイル	- コンパイル結果のオブジェクト・コードの ASCII イメージ・ファイル	asm
	プリプロセス・リスト・ファイル	- #include などのプリプロセス命令を処理した結果のリスト・ファイル - ASCII イメージ・ファイル	ppl
	クロスリファレンス・リスト・ファイル	- C ソース・モジュール・ファイル中で使用されている関数名，変数名の情報が入ったリスト・ファイル	xrf
	エラー・リスト・ファイル	- ソース・ファイルとコンパイル・エラー・メッセージを内容とするリスト・ファイル	ecc cer her er 注
入出力ファイル	テンポラリ・ファイル	- コンパイルのための中間ファイル - コンパイル正常終了時には正式な名前にリネームされ，異常終了時には削除されます。	$nn （ファイル名固定）

注 エラー・リスト・ファイルには，次の 4 通りのファイル・タイプがあります。

ファイル・タイプ	説明
cer	*.c' ファイルに対する，C ソース付きエラー・リスト・ファイル （-se オプション指定で出力）
her	*.h' ファイルに対する，C ソース付きエラー・リスト・ファイル （-se オプション指定で出力）
er	上記以外のファイルに対する，C ソース付きエラー・リスト・ファイル （-se オプション指定で出力）
ecc	すべてのソース・ファイルに対する，C ソースなしエラー・リスト・ファイル （-e オプション指定で出力）

図 3-22 C コンパイラの入出力ファイル



備考 コンパイル・エラーがある場合，エラー・リスト・ファイル，クロスリファレンス・リスト・ファイル以外の各種ファイルは出力されません。

テンポラリ・ファイルは，コンパイル正常終了時に正式な名前にリネームされます。また，異常終了時には削除されます。

3.5 実行開始メッセージ，終了メッセージ

3.5.1 実行開始メッセージ

CC78K0R が起動すると，コンソールに実行開始メッセージが表示されます。

```
78K0R Series C Compiler Vx.xx  [ xx xxx xxxx ]
     Copyright (C) NEC Electronics Corporation xxxx, xxxx
```

3.5.2 実行終了メッセージ

コンパイルの結果，コンパイル・エラーが検出されなかった場合，CC78K0R は，次のメッセージをコンソールに出力して制御を OS に戻します。

```
Target chip : uPD78F1166_A0
Device file : Vx.xx

Compilation complete, 0 error(s) and 0 warning(s) found.
```

コンパイルの結果，コンパイル・エラーが検出された場合，CC78K0R は，エラー・メッセージとエラーの数をコンソールに出力して制御を OS に戻します。

```
prime.c(18) : CC78K0R warning W0745 : Expected function prototype
prime.c(20) : CC78K0R warning W0745 : Expected function prototype
prime.c(26) : CC78K0R warning W0622 : No return value
prime.c(37) : CC78K0R warning W0622 : No return value
prime.c(44) : CC78K0R warning W0622 : No return value

Target chip : uPD78F1166_A0
Device file : Vx.xx

Compilation complete, 0 error(s) and 5 warning(s) found.
```

コンパイル中に処理継続が不可能な致命的エラーが検出された場合，CC78K0R はメッセージをコンソールに出力してコンパイルを中止し，制御を OS に戻します。

次に，エラーが出力された例を示します。

```
78K0R Series C Compiler Vx.xx  [ xx xxx xxxx ]
    Copyright (C) NEC Electronics Corporation xxxx, xxxx

CC78K0R error F0018 : Option is not recognized '-s'
Please enter 'CC78K0R --', if you want help messages.
Program aborted.
```

この例では，存在しないコンパイラ・オプション（-s）を入力したためにエラーとなり，コンパイルが中止されました。

CC78K0R がエラー・メッセージを出力してコンパイルを中止した場合には，そのエラー・メッセージの原因を「第 9 章 エラー・メッセージ」で調べて対処してください。

第 4 章 CC78K0R の機能

この章では，CC78K0R を用いた最適化手法，および，ROM 化機能について説明します。

4.1 最適化手法

CC78K0R では，効率のよいオブジェクト・モジュール・ファイルを生成するために，最適化を行います。
サポートする最適化手法を，次に示します。

表 4-1 最適化手法

フェーズ		内容	例
構文解析部	(1)	定数計算のコンパイル時実行	`a = 3 * 5 ;` → `a = 15 ;`
	(2)	論理式の部分評価による真／偽の判定	`0 && ( a || b )` → `0` `1 || ( a && b )` → `1`
	(3)	ポインタ，配列などのオフセット計算	オフセットをコンパイル時に計算します。
コード生成部	(4)	レジスタ管理	使用していないレジスタを有効に利用します。
	(5)	ターゲット CPU の持つ特殊な命令の利用	`a = a + 1 ;` → `inc` 命令を使用します。 配列要素の代入に転送命令を使用します。
	(6)	短い命令の利用	同じ動作をする命令があった場合，バイト数の短い命令を利用します。 `mov a , #0` → `clrb a`
	(7)	ロング・ジャンプ命令のショート・ジャンプ命令への変更	出力された中間コードを再操作して行います。

フェーズ		内容	例
オプティマイザ	(8)	共通部分式の削除	a = b + c ;　　→ a = b + c ; d = b + c + e ;　　d = a + e ;

	(9)	命令のループの外への移動	for (i = 0 ; i < 10 ; i++)
			{
			　　:
			　a = b + c ;
			　　:
			}
			　　↓
			a = b + c ;
			for (i = 0 ; i < 10 ; i++)
			{
			　　:
			}

	(10)	無用命令の削除	a = a ;　　→ 削除 a = b ; のあと，a が参照されない場合→ 削除（a はオートマティック変数）
	(11)	複写の削除	a = b ;　　→ c = b + d ; c = a + d ; これ以降，a が参照されない場合（a はオートマティック変数）。
	(12)	式の演算順序の変更	レジスタに演算結果を残しておいて，有効となる演算を先に実行します。
	(13)	記憶装置の割り付け（テンポラリ変数）	局所的に使われる変数をレジスタに割り付けます。
	(14)	のぞき穴式最適化	特殊パターンの置き換え 例　a * 1 → a , a + 0 → a
	(15)	演算の強さの軽減	例　a * 2 → a + a , a << 1
	(16)	記憶装置の割り付け（レジスタ変数）	アクセスの速いメモリへデータを割り付けます。 例　レジスタ，saddr（-qr 指定時のみ）
	(17)	ジャンプ最適化（-qj オプション）	連続するジャンプ命令を 1 つにまとめるなど。
	(18)	レジスタ割り付け（-qv/-qr/-rd/-rs オプション）	変数を自動的にレジスタに割り付けるなど。

備考　(1) 〜 (7)，(14)，(15) は，最適化オプションの指定によらず行われます。

(8) 〜 (13)，(17)，(18) の最適化は，最適化オプションが指定された場合に行われます。

(16) は，C ソース・プログラム中に register 宣言がある場合に行います。ただし，saddr 領域には，-qr オプション指定時のみ割り付けます。

最適化オプションについては，「第 5 章 コンパイラ・オプション」を参照してください。

4.2 ROM 化機能

ROM 化とは，初期値あり外部変数などの初期値を ROM に配置しておき，システム実行時に RAM にコピーする処理です。

CC78K0R は，プログラムの ROM 化処理付きのスタートアップ・ルーチンを提供しているので，スタートアップ時の ROM 化処理などを記述する手間が省けます。

スタートアップ・ルーチンの内容については，「8.3 スタートアップ・ルーチン」を参照してください。

次に，プログラムの ROM 化を行う方法について説明します。

4.2.1 リンク時

リンク時には，スタートアップ・ルーチン，オブジェクト・モジュール・ファイルとライブラリをリンクします。スタートアップ・ルーチンは，オブジェクト・プログラムの初期化を行います。

- s0r*.rel

  スタートアップ・ルーチン（ROM 化対応）です。

  初期化データのコピー・ルーチンを含み，初期化データの開始を示します。スタート・アドレスには，“_@cstart”というラベル（シンボル）が付けられます。

- cl0r*.lib

  CC78K0R に添付されているライブラリです。

  このライブラリ・ファイルの中には，次のものが含まれています。

  (1) ランタイム・ライブラリ

  ランタイム・ライブラリ名は，シンボルの先頭に“@@”が付加されます。ただし，特殊ライブラリ cprep，cdisp には先頭に“_@”が付加されます。

  (2) 標準ライブラリ

  標準ライブラリ名は，シンボルの先頭に“_”が付加されます。

- *.lib

  ユーザ作成のライブラリです。

  シンボルの先頭に“_”が付加されます。

注意 CC78K0R は，何種類かのスタートアップ・ルーチン，およびライブラリを提供しています。スタートアップ・ルーチンについては，「第 8 章 スタートアップ・ルーチン」を参照してください。ライブラリについては「2.5.1 ライブラリ・ファイル」を参照してください。

第 5 章　コンパイラ・オプション

この章では，CC78K0R のコンパイラ・オプションについて説明します。

CC78K0R を起動する際に，コンパイラ・オプションを指定することができます。コンパイラ・オプションは，CC78K0R の動作に対して指示を与えたり，プログラムの実行に先立って必要な情報を指示するためのものです。

コンパイラ・オプションは，単一ではなく，複数を同時に指定することができます。ユーザは，目的に合わせてコンパイラ・オプションを選択し，効率のよい作業を行うことができます。

5.1　指定方法

コンパイラ・オプションは，次の方法で指定することができます。

- CC78K0R を起動するときに，コマンド行で指定します。
- PM+ の [コンパイラオプションの設定] ダイアログで指定します。
- パラメータ・ファイル内で指定します。

上記コンパイラ・オプションの指定方法例については，「第 3 章 コンパイルからリンクまでの手順」を参照してください。

また，コンパイラ・オプションに続くサブオプション，ファイル名などは，スペースなどの空白を入れずに続けて指定してください。コンパイラ・オプション間は，空白が必要です。

なお，コンパイラ・オプションに大文字，小文字の区別はありません。

＜例＞

```
cc78k0r △ -cf1166a0 △ prime.c △ -aprime.asm △ -qx2
```

備考　△：スペースなどの空白

5.2 優先度

次の表に示すコンパイラ・オプションのうち，縦軸のものと横軸のものを同時に 2 つ以上指定した場合の優先度について説明します。

表 5-1 コンパイラ・オプションの優先度

	-no	-p	-np	-d	-u	-a	-e	-x	-sa
-r	×	—	—	—	—	—	—	—	—
-q	×	—	—	—	—	—	—	—	—
-g	×	—	—	—	—	—	—	—	—
-k	—	△	×	—	—	—	—	—	—
-d	—	—	—	—	○	—	—	—	—
-u	—	—	—	○	—	—	—	—	—
-sa	—	—	—	—	—	×	—	—	—
-lw	—	△	—	—	—	△	△	△	—
-ll	—	△	—	—	—	△	△	△	—
-lt	—	△	—	—	—	△	△	△	—
-lf	—	△	—	—	—	△	△	△	—
-li	—	—	—	—	—	—	—	—	△

[×で記した箇所]

横軸に示したオプションを指定した場合，縦軸に示したオプションは無効となります。

＜例＞

```
C>cc78k0r -cf1166a0 -e sample.c -no -rd -g
```

-rd，-g オプションは，無効となります。

[△で記した箇所]

横軸に示したオプションを指定しない場合，縦軸のオプションは無効となります。

＜例＞

```
C>cc78k0r -cf1166a0 -e sample.c -p -k
```

-p オプションが指定されているので，-k オプションは有効です。

[○で記した箇所]

横軸のオプションと縦軸のオプションで，後ろに指定したものが優先となります。

＜例＞

```
C>cc78k0r -cf1166a0 -e sample.c -utest -dtest=1
```

-d オプションが後ろに指定されているので，-u オプションは無効となり，-d オプションが優先されます。

また，-o/-no オプションのように，オプション名の前に n を付加できるオプションでも，後ろに指定したものが優先となります。

＜例＞

```
C>cc78k0r -cf1166a0 -e sample.c -o -no
```

-no オプションが後ろに指定されているので，-o オプションは無効となり，-no オプションが優先されます。

表 5-1 に記述されていないオプションは，他のオプションの影響を特に受けません。しかし，ヘルプ指定オプション（--/-?/-h）が指定された場合には，すべてのオプション指定が無効となります。

なお，ヘルプ指定オプション（--/-?/-h）は，PM+ 上では指定することができません。PM+ 上でヘルプを参照する場合は，PM+ の各オプション・ダイアログで [ヘルプ] ボタンを押してください。

5.3 種類

コンパイラ・オプションは，次のオプションに分類することができます。

表 5-2 コンパイラ・オプション一覧

分類	オプション	説明
デバイス種別指定	-c	対象とするデバイス種別を指定します。
オブジェクト・モジュール・ファイル作成指定	-o	オブジェクト・モジュール・ファイルの出力を指定します。
	-no	
メモリ配置指定	-r	メモリの割り付け方法を指定します。
	-nr	
	-rd	外部変数／外部スタティック変数を自動的に saddr 領域に割り付けるように指定します。
	-nr	
	-rs	static auto 変数を自動的に saddr 領域に割り付けるように指定します。
	-nr	
最適化指定	-q	最適化種別を指定します。
	-nq	
デバッグ情報出力指定	-g	C ソース・レベルのデバッグ情報の出力を指定します。
	-ng	
プリプロセス・リスト・ファイル作成指定	-p	プリプロセス・リスト・ファイルの出力を指定します。
	-k	プリプロセス・リストに対する処理を指定します。
プリプロセス指定	-d	マクロ定義を行います。
	-u	マクロ定義を無効にします。
	-i	インクルード・ファイルを指定したフォルダから読み込みます。
アセンブラ・ソース・モジュール・ファイル作成指定	-a	アセンブラ・ソース・モジュール・ファイルの出力を指定します。
	-sa	
エラー・リスト・ファイル作成指定	-e	エラー・リスト・ファイルの出力を指定します。
	-se	
クロスリファレンス・リスト・ファイル作成指定	-x	クロスリファレンス・リスト・ファイルの出力を指定します。

分類	オプション	説明
リスト形式指定	-lw	各種リスト・ファイルの 1 行の文字数を指定します。
	-ll	各種リスト・ファイルの 1 ページの行数を指定します。
	-lt	各種リスト・ファイルのタブの展開文字数を変更します。
	-lf	各種リスト・ファイルの最後に改ページ・コードを付加します。
	-li	C ソース付きアセンブラ・ソース・モジュール・ファイルにインクルード・ファイルの C ソースも付加します。
ワーニング出力指定	-w	ワーニング・メッセージをコンソールへ出力するか否かを指定します。
実行状態表示指定	-v	コンパイルの実行状態をコンソールに出力するか否かを指定します。
	-nv	
パラメータ・ファイル指定	-f	入力ファイル名，およびオプションを，指定したファイルより入力します。
テンポラリ・ファイル作成フォルダ指定	-t	テンポラリ・ファイルを指定したドライブ，フォルダに作成します。
ヘルプ指定	--	コンソールにヘルプ・メッセージを出力します。
	-?	
	-h	
機能拡張指定	-z	拡張機能に対する処理を有効にします。
	-nz	
デバイス・ファイルのサーチ・パス	-y	デバイス・ファイルをサーチするパスを指定します。
メモリ・モデル指定	-m	コンパイル時のメモリ・モデルの種類を指定します。

5.4 説明

以降に，各コンパイラ・オプションの詳細を説明します。

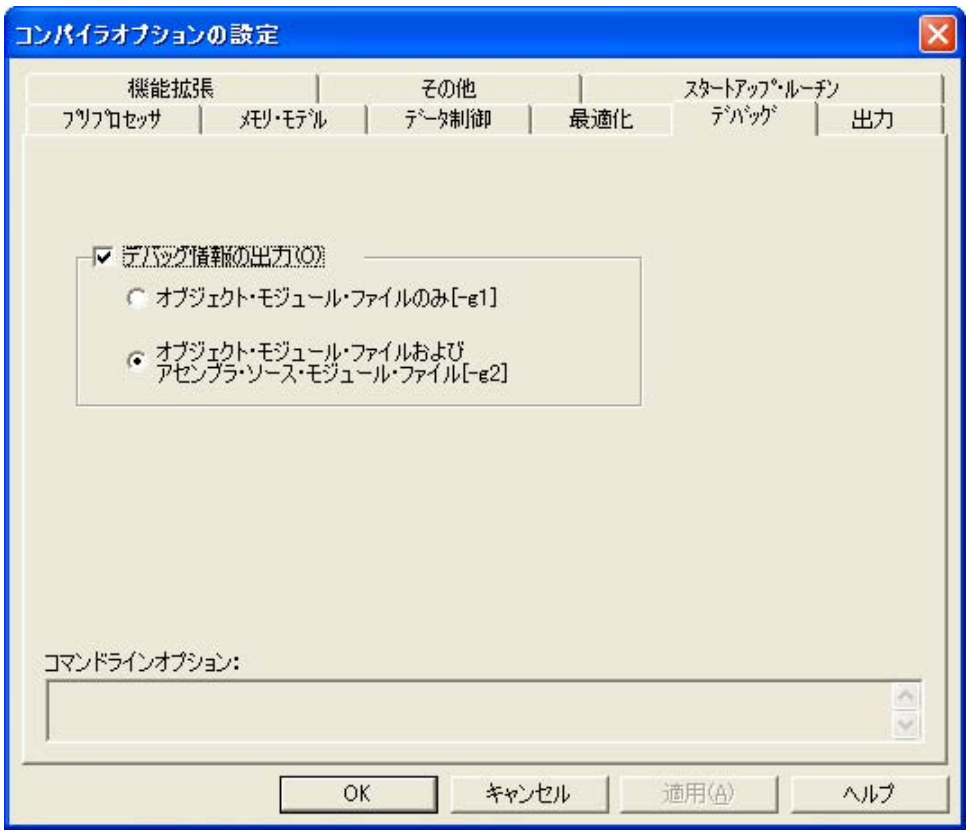
ここでの使用例は，コマンド・ライン上で CC78K0R を起動した場合の例です。PM+ 上で起動する場合は，コマンドとデバイス種別指定と C ソースを除いたオプションを [コンパイラオプションの設定] ダイアログで指定してください。

【例：コマンド・ライン】

```
C>cc78k0r -cf1166a0 prime.c -g
```

【例：PM+ 使用時】

図 5-1 [コンパイラオプションの設定] ダイアログ



デバイス種別指定

-C

(1) -c

【記述形式】

```
-c デバイス種別
```

- 省略時解釈

  省略することはできません。

【機能】

- -c オプションは，コンパイルの対象となるターゲット・デバイスを指示します。

【用途】

- 必ず指定してください。CC78K0R は，指定されたターゲット・デバイスに対してコンパイルを行い，それに対応したオブジェクト・コードを生成します。

【説明】

- -c オプションで指定可能なターゲット・デバイスと，それに対応するデバイス種別に関しては，各デバイスのユーザーズ・マニュアル，または「デバイス・ファイル　使用上の留意点」を参照してください。
- CC78K0R 使用時には，デバイス・ファイルが必要となります。

【注意】

- -c オプションを省略することはできません。ただし，C ソース中に次の記述がある場合には，コマンド行での指定を省略することができます。

```
#pragma pc ( デバイス種別 )
```

- C ソース中とコマンド行で異なるデバイス指定をした場合，コマンド行のデバイスが優先されます。
- PM+ 使用時は，このオプションの設定はプロジェクト設定により決定されるため，コンパイラ・オプションで指定する必要はありません。

【使用例】

- コマンド行でターゲット・デバイス uPD78F1166_A0 を指定します。

```
C>cc78k0r -cf1166a0 prime.c
```

- C ソース prime.c 中でターゲット・デバイス uPD78F1166_A0 を指定して，起動させます。

```
#pragma pc ( f1166a0 )
#define TRUE    1
#define FALSE   0
#define SIZE    200

char    mark [ SIZE + 1 ] ;

main ( ) {
        int     i , prime , k , count ;
          :
}
```

これにより，ターゲット・デバイスの指定は，コマンド行で省略することができます。

```
C>cc78k0r prime.c
```

- C ソース prime.c 中とコマンド行で異なるデバイスを指定し，起動させます。

＜C ソース＞

```
#pragma pc ( f1166a0 )
#define TRUE    1
#define FALSE   0
#define SIZE    200

char    mark [ SIZE + 1 ] ;

main ( ) {
        int     i , prime , k , count ;
          :
}
```

＜コマンド・ライン＞

```
C>cc78k0r -cf1176 prime.c
```

コマンド行で指定したターゲット・デバイスの指定が優先され，次のように C コンパイラが実行されます。

```
78K0R Series C Compiler Vx.xx  [ xx xxx xxxx ]
    Copyright (C) NEC Electronics Corporation xxxx, xxxx

sample¥prime.c ( 1 )  : CC78K0R warning W0832 : Duplicated chip specifier
sample¥prime.c ( 18 ) : CC78K0R warning W0745 : Expected function prototype
sample¥prime.c ( 20 ) : CC78K0R warning W0745 : Expected function prototype
sample¥prime.c ( 26 ) : CC78K0R warning W0622 : No return value
sample¥prime.c ( 37 ) : CC78K0R warning W0622 : No return value
sample¥prime.c ( 44 ) : CC78K0R warning W0622 : No return value

Target chip : uPD78F1176
Device file : Vx.xx

Compilation complete, 0 error(s) and 6 warning(s) found.
```

オブジェクト・モジュール・ファイル作成指定

-o/-no

(1) -o/-no

【記述形式】

```
-o[ 出力ファイル名 ]
-no
```

- 省略時解釈

  -o *入力ファイル名* .rel

【機能】

- -o オプションは，オブジェクト・モジュール・ファイルの出力を指示します。また，その出力先や出力ファイル名を指示します。
- -no オプションは，オブジェクト・モジュール・ファイルを出力しません。

【用途】

- オブジェクト・モジュール・ファイルの出力先や出力ファイル名を変更したいときに，-o オプションを指定します。
- アセンブラ・ソース・モジュール・ファイルの出力のみが目的でコンパイルする場合などに，-no オプションを指定します。これにより，コンパイル時間が短縮されます。

【説明】

- -o オプションを指定する際に出力ファイル名を省略すると，オブジェクト・モジュール・ファイル名は，“*入力ファイル名* .rel”となります。
- -o オプションを指定する際に出力ファイル名の拡張子を省略すると，オブジェクト・モジュール・ファイル名は，“*出力ファイル名* .rel”となります。
- -o オプションを指定しても，コンパイル・エラーがある場合は，オブジェクト・モジュール・ファイルは出力されません。
- -o オプションを指定する際にドライブ名を省略すると，カレント・ドライブにオブジェクト・モジュール・ファイルが出力されます。
- -o と -no の両オプションが同時に指定された場合は，後ろに指定したものが優先となります。

【注意】

- PM+ 使用時に，出力先を変更する場合は，[出力] タブの“オブジェクト・モジュール・ファイルの出力 (O)”の“出力パス”コンボ・ボックスに，出力先を指定してください。
- 個別コンパイラ・オプション設定時は，出力ファイル名の変更も可能です。
- [出力] タブの“出力パス”コンボ・ボックスに，ファイル名，または出力先を指定してください。

【使用例】

- 先に指定した -no オプションは無効，後ろに指定した -o オプションは有効となり，オブジェクト・モジュール・ファイル prime.o が出力されます。

```
C>cc78k0r -cf1166a0 prime.c -no -o
```

メモリ配置指定

-r/-nr，-rd/-nr，-rs/-nr

(1) -r/-nr

【記述形式】

```
-r 処理種別（複数指定可能）
-nr
```

- 省略時解釈

  -nr

【機能】

- -r オプションは，メモリへの割り付け方法を指示します。
- -nr オプションは，-r オプションを無効にします。

【用途】

- プログラムをメモリへ割り付けるときの，割り付け方法を指定します。

【説明】

- -r オプションで指定可能な処理種別を，次に示します。

  処理種別の指定を省略することはできません。省略した場合，致命的エラー（F0012）となります。

処理種別	内容
a	間接参照を 1 バイト単位で行います。
b	ビット・フィールドを MSB から割り付けます。
d[*n*][m] （*n* = 1, 2, 4）	外部変数／外部スタティック変数（const 型を除く）を sreg 宣言の有無にかかわらず，自動的に saddr 領域に割り付けます。 詳細については，「(2) -rd/-nr」を参照してください。
s[*n*][m] （*n* = 1, 2, 4）	static auto 変数を sreg 宣言の有無にかかわらず，自動的に saddr 領域に割り付けます。 詳細については，「(3) -rs/-nr」を参照してください。
c	間接参照を 1 バイト単位で行います。 さらに，構造体をパッキングし，構造体メンバを 1 バイト・アラインで配置します。

備考　処理種別は，複数指定することができます。

- -nr オプションが指定された場合は，次のように解釈されます。

処理種別	内容
a	間接参照を 1 バイト単位で行いません。
b	ビット・フィールドを LSB から割り付けます。
d	saddr 領域に自動的に割り付けません。
s	saddr 領域に自動的に割り付けません。
c	間接参照を 1 バイト単位で行いません。 さらに，構造体をパッキングせず，構造体メンバを 1 バイト・アラインで配置しません。

【使用例】

- 外部変数／外部スタティック変数，static auto 変数を sreg 宣言の有無にかかわらず，自動的に saddr 領域に割り付けます。

```
C>cc78k0r -cf1166a0 -rds
```

(2) -rd/-nr

【記述形式】

```
-rd[n][m]（n = 1, 2, 4）
-nr
```

- 省略時解釈

  -nr

【機能】

- -rd オプションは，外部変数／外部スタティック変数を自動的に saddr 領域に割り付けるように指示します。
- -nr オプションは，-rd オプションを無効にします。

【用途】

- 外部変数／外部スタティック変数（const 型を除く）を sreg 宣言の有無にかかわらず，自動的に saddr 領域に割り付けます。

【説明】

- *n* の値と m の指定によって，割り付ける変数の最大幅を次のように指定します。

n，m の指定	saddr 領域に割り当てる変数
n	- *n* = 1 の場合 char，unsigned char - *n* = 2 の場合 char，unsigned char，short，unsigned short，int，unsigned int，enum，near ポインタ - *n* = 4 の場合 char，unsigned char，short，unsigned short，int，unsigned int，enum，long，unsigned long，ポインタ
m	構造体，共用体，配列
省略した場合	すべての変数

- sreg 宣言された変数は，-rd オプションの指定にかかわらず，saddr 領域に割り付けられます。
- extern 宣言により参照する変数については，saddr 領域に割り付けられるものとして処理されます。
- このオプションにより saddr 領域に割り付けられた変数は，sreg 変数と同様に扱います。

【使用例】

- char，unsigned char 型の外部変数／外部スタティック変数を sreg 宣言の有無にかかわらず，自動的に saddr 領域に割り付けます。

```
C>cc78k0r -cf1166a0 -rd1
```

(3) -rs/-nr

【記述形式】

```
-rs[n][m]（n = 1, 2, 4）
-nr
```

- 省略時解釈

  -nr

【機能】

- -rs オプションは，static auto 変数を自動的に saddr 領域に割り付けるように指示します。
- -nr オプションは，-rs オプションを無効にします。

【用途】

- static auto 変数を sreg 宣言の有無にかかわらず，自動的に saddr 領域に割り付けます。

【説明】

- *n* の値と m の指定によって，割り付ける変数の最大幅を次のように指定します。

n，m の指定	saddr 領域に割り当てる変数
n	- *n* = 1 の場合 char，unsigned char - *n* = 2 の場合 char，unsigned char，short，unsigned short，int，unsigned int，enum，near ポインタ - *n* = 4 の場合 char，unsigned char，short，unsigned short，int，unsigned int，enum，long，unsigned long，ポインタ
m	構造体，共用体，配列
省略した場合	すべての変数

- sreg 宣言された変数は，-rs オプションの指定にかかわらず，saddr 領域に割り付けられます。
- このオプションにより saddr 領域に割り付けられた変数は，sreg 宣言された static auto 変数と同様に扱います。

【使用例】

- char，unsigned char 型の static auto 変数を sreg 宣言の有無にかかわらず，自動的に saddr 領域に割り付けます。

```
C>cc78k0r -cf1166a0 -rs1
```

最適化指定

-q/-nq

(1) -q/-nq

【記述形式】

```
-q[ 最適化種別 ]（複数指定可能）
-nq
```

- 省略時解釈

  -qcjlvw

【機能】

- -q オプションは，最適化フェーズを呼び出し効率のよいオブジェクトを生成するよう指示します。
- -nq オプションは，-q オプションを無効にします。

【用途】

- オブジェクトの実行速度の向上，コード・サイズを削減したい場合に，-q オプションを指定します。

  -q オプション指定時に，複数の最適化を同時に有効にしたい場合は，最適化種別を続けて指定します。詳細については，**【説明】**を参照してください。

【説明】

- -q オプションで指定可能な最適化種別を，次に示します。

<table>
<tr><th>最適化種別</th><th>処理内容</th></tr>
<tr><td>省略</td><td>-qcjlvw が指定されたと見なします。</td></tr>
<tr><td>u</td><td>修飾子なしの char を unsigned char と見なすことにより，コード効率を良くします。</td></tr>
<tr><td>c</td><td>char に関する演算を符号拡張せずに行います。
<table><tr><th>演算対象</th><th>演算結果</th></tr><tr><td>unsigned char 型の変数と unsigned char 型の変数</td><td>unsigned char 型</td></tr><tr><td>unsigned char 型の変数と signed char 型の変数</td><td>unsigned char 型</td></tr><tr><td>signed char 型の変数と signed char 型の変数</td><td>signed char 型</td></tr><tr><td>-128 ～ 255 の定数と unsigned char 型の変数</td><td>unsigned char 型</td></tr><tr><td>-128 ～ 127 の定数と signed char 型の変数</td><td>signed char 型</td></tr><tr><td>0 ～ 255 で接尾語 U 付きの定数と signed char 型の変数</td><td>unsigned char 型</td></tr></table></td></tr>
<tr><td>r[n]
（n = 1, 2）</td><td>レジスタ変数をレジスタに加えて saddr 領域にも割り当てます。
-ql2 を同時に指定することにより，コード・サイズを削減することができる可能性があります。
n の値によって，レジスタ変数を割り当てる範囲が次のように異なります。n を省略した場合は，n = 2 として解釈されます。
1 ： norec の引数と auto 変数を saddr 領域に割り当てる
2 ： norec の引数，auto 変数，および register 変数を saddr 領域に割り当てる</td></tr>
<tr><td>j</td><td>分岐命令の最適化を行います。</td></tr>
<tr><td>x[n]
（n = 1 - 2）</td><td>最適化オプションを，スピード／コード・サイズの優先順位により自動的に割り当てます。
n の値によって，割り当てるオプションが次のように異なります。n を省略した場合は，n = 2 として解釈されます。
1 ： スピード優先として，-qcjvw を指定したものと見なす
2 ： デフォルトとして，-qcjlvw を指定したものと見なす</td></tr>
<tr><td>w</td><td>積極的な最適化を行います。
演算式の実行順序の入れ替えを行います。</td></tr>
<tr><td>v</td><td>引数，およびオートマティック変数を，レジスタ，または saddr 領域に自動的に割り当てます。</td></tr>
<tr><td>l[n]
（n = 1 - 2）</td><td>コード・サイズを優先した最適化を行い，定型コード・パターンをライブラリに置き換えます。指定しない場合は，スピード優先の最適化を行います。
n の値によって，ライブラリに置き換える範囲が次のように異なります。n を省略した場合は，n = 1 として解釈されます。
1 ： ライブラリに置き換えない
2 ： 関数の前後処理のみ</td></tr>
</table>

- 最適化種別は複数指定することができます。
- -q オプションを省略した場合，または最適化種別を省略した場合，-qcjlvw のオプションを指定した場合と同じ最適化を行います。
- デフォルト・オプションの一部を解除するには，解除したいオプション以外のオプションを指定することによって解除することができます（例：-qr を指定 → -qcjlvw を解除）。
- オブジェクト・モジュール・ファイル，アセンブラ・ソース・モジュール・ファイルのいずれも出力されない場合，-qu 以外の -q オプションは無効となります。
- -q と -nq の両オプションが同時に指定された場合は，後ろに指定したものが有効になります。
- -q オプションが同時に複数指定された場合，最後に指定したものが有効となります。

【使用例】

- 修飾子なしの char を unsigned と見なすことにより，コード効率を良くします。

```
C>cc78k0r -cf1166a0 prime.c -qu
```

- 先に指定した -qc オプションは無効，後ろに指定した -qr オプションは有効となり，norec の引数，auto 変数，および register 変数を saddr 領域に割り当てます。

```
C>cc78k0r -cf1166a0 prime.c -qc -qr
```

- -qc と -qr の両方のオプションを有効にしたい場合は，次のようにコマンド入力します。

```
C>cc78k0r -cf1166a0 prime.c -qcr
```

デバッグ情報出力指定

-g/-ng

(1) -g/-ng

【記述形式】

```
-g[n]（n = 1, 2）
-ng
```

- 省略時解釈

  -g2

【機能】

- -g オプションは，オブジェクト・モジュール・ファイル中にデバッグ情報を付加するよう指示します。
- -ng オプションは，-g オプションを無効にします。

【用途】

- -g オプションを指定しない場合，デバッガへの入力となるオブジェクト・モジュール・ファイルに必要な行番号，シンボル情報が出力されません。したがって，ソース・レベル・デバッグを行うときには，リンクするすべてのモジュールを -g オプション指定でコンパイルします。

【説明】

- *n* の値による動作の違いを次に示します。

n の値	内容
省略	*n* = 2 が指定されたものと見なします。
1	オブジェクト・モジュール・ファイル中のみデバッグ情報（$DGS，$DGL で始まる情報）を付加し，アセンブラ・ソース・モジュール・ファイル中には付加しません。 このオプションは，アセンブラ・ファイルを参照しやすくするためのものです。 オブジェクト・モジュール・ファイルには，デバッグ情報が付加されるため，ソース・デバッグも可能です。
2	オブジェクト・モジュール・ファイル中，アセンブラ・ソース・モジュール・ファイル中にデバッグ情報を付加します

- -g と -ng が同時に指定された場合は，後ろに指定したものが有効となります。
- オブジェクト・モジュール・ファイル，アセンブラ・ソース・モジュール・ファイルのいずれも出力されない場合，-g オプションは無効となります。

【使用例】

- オブジェクト・モジュール・ファイル prime.o 中にデバッグ情報を付加します。

```
C>cc78k0r -cf1166a0 prime.c -g
```

プリプロセス・リスト・ファイル作成指定

-p，-k

(1) -p

【記述形式】

-p[*出力ファイル名*]

- 省略時解釈

  なし（ファイルを出力しません）

【機能】

- -p オプションは，プリプロセス・リスト・ファイルを出力することを指示します。また，その出力先や出力ファイル名を指示します。-p オプションが省略された場合，プリプロセス・リスト・ファイルを出力しません。

【用途】

- プリプロセス処理を -k オプションの処理種別に従って行ったあとのソース・ファイルを出力させたいとき，あるいは，プリプロセス・リスト・ファイルの出力先や出力ファイル名を変更したいときに，-p オプションを指定します。

【説明】

- -p オプションを指定する際に出力ファイル名を省略すると，プリプロセス・リスト・ファイル名は，“*入力ファイル名* .ppl”となります。
- -p オプションを指定する際に出力ファイル名の拡張子を省略すると，プリプロセス・リスト・ファイル名は，“*出力ファイル名* .ppl”となります。
- -p オプションを指定する際にドライブ名を省略すると，カレント・ドライブにプリプロセス・リスト・ファイルが出力されます。

【注意】

- PM+ 使用時に，出力先を変更する場合は，[出力] タブの“プリプロセス・リスト・ファイルの出力 (E)”の“出力パス”コンボ・ボックスに，出力先を指定してください。
- 個別コンパイラ・オプション設定時は，出力ファイル名の変更も可能です。
- [出力] タブの“出力パス”コンボ・ボックスに，ファイル名，または出力先を指定してください。

【使用例】

- プリプロセス・リスト・ファイル sample.ppl を出力します。

```
C>cc78k0r -cf1166a0 prime.c -psample.ppl
```

(2) -k

【記述形式】

-k[*処理種別*]（複数指定可能）

- 省略時解釈

  -kfln

【機能】

- -k オプションは，プリプロセス・リストに対する処理を指示します。

【用途】

- プリプロセス・リスト・ファイルを出力する際にコメントを削除したり，定義の展開を参照するときに指定します。

【説明】

- -k オプションで指定可能な処理種別を次に示します。

処理種別	内容
省略	-kfln が指定されたものと見なします。
c	コメントの削除
d	#define の展開
f	#if，#ifdef，#ifndef の条件コンパイル
i	#include の展開
l	#line の処理
n	行番号とページング処理

備考　処理種別は，複数指定することができます。

- -p オプションが指定されない場合は，-k オプションは無効となります。
- -k オプションが同時にいくつか指定された場合は，最後に指定したものが有効となります。

【使用例】

- プリプロセス・リスト・ファイル prime.ppl を出力する際，コメントの削除，行番号とページング処理を行います。

```
C>cc78k0r -cf1166a0 prime.c -p -kcn
```

＜出力例＞

```
/*
78K0R Series C Compiler Vx.xx Preprocess List
                                        Date : XX XXX XXXX Page :   1

Command         : -cf1166a0 prime.c -p -kcn
In-file         : prime.c
PPL-file        : prime.ppl
Para-file       :
*/

        1 : #define TRUE    1
        2 : #define FALSE   0
        3 : #define SIZE    200
        4 :
        5 : char    mark [ SIZE + 1 ] ;
        6 :
        7 : main ( )
        8 : {
         :
/*
Target chip     : uPD78F1166_A0
Device file     : Vx.xx
*/
```

プリプロセス指定

-d，-u，-i

(1) -d

【記述形式】

```
-d マクロ名 [= 定義名 ][, マクロ名 [= 定義名 ]] …（複数指定可能）
```

- 省略時解釈

  C ソース・モジュール・ファイル中のマクロ定義のみを有効とします。

【機能】

- -d オプションは，C ソース中の #define 文と同様のマクロ定義を行うよう指示します。

【用途】

- 特定のマクロ定義を有効にしたいときに指定します。

【説明】

- 各定義を“,”で区切ることにより，一度に 30 個までマクロ定義を行うことができます。
- “=”と“,”の前後には，空白を入れることはできません。
- 定義名が省略されると，“マクロ名 =1”として扱われます。
- -d と -u の両オプションで同じマクロ名が指定された場合，後ろに指定したものが有効となります。

【使用例】

- C ソース prime.c 中に，

  #define TEST　1

  #define TIME　10

  の定義が行われていたものとします。

```
C>cc78k0r -cf1166a0 prime.c -dTEST,TIME=10
```

(2) -u

【記述形式】

```
-u マクロ名 [, マクロ名 ] …（複数指定可能）
```

- 省略時解釈

  -d で指定したマクロ定義を有効とします。

【機能】

- -u オプションは，C ソース中の #undef 文と同様にマクロ定義を無効にします。

【用途】

- -d オプションで定義されたマクロ名を無効にするときに指定します。

【説明】

- 各マクロ名を“,”で区切ることにより，1 度に 30 個までマクロ定義を無効にすることができます。
  なお，“,”の前後には空白を入れることはできません。
- -u オプションで無効にすることができるマクロ定義は，-d オプションで定義されたものです。
  C ソース・モジュール・ファイル中に #define によって定義されたマクロ名や，CC78K0R の持つシステム・マクロ名は，-u オプションで無効にすることはできません。
- -d と -u の両オプションで同じマクロ名が指定された場合は，後ろに指定したものが有効となります。

【使用例】

- 先に指定した -d オプションは無効，後ろに指定した -u オプションは有効となり，TEST のマクロ定義を無効にしています。

```
C>cc78k0r -cf1166a0 prime.c -dTEST,TIME=10 -uTEST
```

(3) -i

【記述形式】

-i *フォルダ*[, *フォルダ*] …（複数指定可能）

- 省略時解釈

  下記フォルダが指定されたものとして扱われます。

  (1) ソース・ファイルのあるフォルダ[注 1]

  (2) 環境変数 INC78K0R により指定されたフォルダ

  (3) C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx*¥inc78k0r [注 2]

【機能】

- -i オプションは，C ソース中の #include 文で指定されたインクルード・ファイルを指定されたフォルダから入力するよう指示します。

【用途】

- インクルード・ファイルをあるフォルダから検索したいときに指定します。

【説明】

- “ , ”で区切ることにより，一度に 64 個までフォルダを指定することができます。
  なお，“ , ”の前後には空白を入れることはできません。
- -i に続いてフォルダが複数指定されるか，あるいは -i オプションが複数指定された場合，指定された順番に #include で指定したファイルを検索します。
- 検索順序は次のようになります。

  (1) ソース・ファイルのあるフォルダ[注 1]

  (2) -i オプションにより指定されたフォルダ

  (3) 環境変数 INC78K0R により指定されたフォルダ

  (4) C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx*¥inc78k0r [注 2]

注 1 #include 文で，インクルード・ファイル名を " "（ダブル・コーテーション）で指定した場合は，ソース・ファイルのあるフォルダを最初に検索します。< > で指定した場合は，検索されません。

注 2 C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx* にインストールした場合の例です。

【使用例】

- C ソース prime.c 中の #include 文で指定されたインクルード・ファイルをフォルダ b:，および b:¥sample から入力します。

```
C>cc78k0r -cf1166a0 prime.c -ib:,b:¥sample
```

アセンブラ・ソース・モジュール・ファイル作成指定

-a，-sa

(1) -a

【記述形式】

-a[*出力ファイル名*]

- 省略時解釈

  アセンブラ・ソース・モジュール・ファイルを出力しません。

【機能】

- -a オプションは，アセンブラ・ソース・モジュール・ファイルの出力を指示します。また，その出力先や出力ファイル名を指示します。

【用途】

- アセンブラ・ソース・モジュール・ファイルの出力先や出力ファイル名を変更したいときに，-a オプションを指定します。

【説明】

- ファイル名としてディスク型ファイル名とデバイス型ファイル名を指定することができます。
- -a オプションを指定する際に出力ファイル名を省略すると，アセンブラ・ソース・モジュール・ファイル名は，“*入力ファイル名* .asm”となります。
- -a オプションを指定する際に出力ファイル名の拡張子を省略すると，アセンブラ・ソース・モジュール・ファイル名は，“*出力ファイル名* .asm”となります。
- -a オプションを指定する際にドライブ名を省略すると，カレント・ドライブにアセンブラ・ソース・モジュール・ファイルが出力されます。
- -a と -sa の両オプションが同時に指定された場合は，-sa オプションは無視されます。

【注意】

- PM+ 使用時に，出力先を変更する場合は，[出力] タブの“アセンブラ・ソース・モジュール・ファイルの出力 (C)”の“出力パス”コンボ・ボックスに出力先を指定し，“C ソース無し [-a]”を選択してください。
- 個別コンパイラ・オプション設定時は，出力ファイル名の変更も可能です。
- [出力] タブの“出力パス”コンボ・ボックスに，ファイル名，または出力先を指定してください。ソース・ファイル名として指定する場合は，拡張子は“asm” にしてください。

【使用例】

- アセンブラ・ソース・モジュール・ファイル sample.asm を出力します。

```
C>cc78k0r -cf1166a0 prime.c -asample.asm
```

(2) -sa

【記述形式】

-sa[*出力ファイル名*]

- 省略時解釈

  アセンブラ・ソース・モジュール・ファイルを出力しません。

【機能】

- -sa オプションは，アセンブラ・ソース・モジュール・ファイルにコメントとして C ソースを付加します。また，その出力先や出力ファイル名を指示します。

【用途】

- アセンブラ・ソース・モジュール・ファイルと C ソース・モジュール・ファイルを一緒に出力したいときに，-sa オプションを指定します。

【説明】

- ファイル名として，ディスク型ファイル名とデバイス型ファイル名を指定することができます。
- -sa オプションを指定する際に出力ファイル名を省略すると，アセンブラ・ソース・モジュール・ファイル名は，“*入力ファイル名* .asm”となります。
- -sa オプションを指定する際に出力ファイル名の拡張子を省略すると，アセンブラ・ソース・モジュール・ファイル名は，“*出力ファイル名* .asm”となります。
- -sa オプションを指定する際にドライブ名を省略すると，カレント・ドライブにアセンブラ・ソース・モジュール・ファイルが出力されます。
- -sa と -a の両オプションが同時に指定された場合は，-sa オプションは無視されます。
- 出力アセンブラ・ソース・モジュールのコメントには，インクルード・ファイルの C ソースは付加しません。ただし，-li オプションを指定した場合は，コメントにインクルード・ファイルの C ソースも付加します。

【注意】

- PM+ 使用時に，出力先を変更する場合は，[出力] タブの“アセンブラ・ソース・モジュール・ファイルの出力”の“出力パス”コンボ・ボックスに出力先を指定し，“C ソース付き”を選択してください。
- 個別コンパイラ・オプション設定時は，出力ファイル名の変更も可能です。
- [出力] タブの“出力ファイル”コンボ・ボックスに，ファイル名，または出力先を指定してください。ソース・ファイル名として指定する場合は，拡張子は“asm” にしてください。

【使用例】

- アセンブラ・ソース・モジュール・ファイル prime.asm にコメントとして C ソース prime.c を付加します。

```
C>cc78k0r -cf1166a0 prime.c -sa
```

＜出力例＞

```
; 78K0R Series C Compiler Vx.xx Assembler Source
;                                     Date : xx xxx xxxx Time : xx : xx : xx

; Command       : -cf1166a0 prime.c -sa
; In-file       : prime.c
; Asm-file      : prime.asm
; Para-file     :

$PROCESSOR ( f1166a0 )
$DEBUG
$NODEBUGA
$KANJICODE SJIS
$TOL_INF        03FH , 100H , 00H , 00H , 00H

$DGS    FIL_NAM , .file ,       037H , 0FFFEH , 03FH , 067H , 01H , 00H
$DGS    AUX_FIL , prime.c
$DGS    MOD_NAM , prime ,       00H , 0FFFEH , 00H , 077H , 00H , 00H
          :
        EXTRN   _@RTARG0
        EXTRN   @@isrem
        PUBLIC  _printf
        PUBLIC  _putchar
        PUBLIC  _mark
        PUBLIC  _main
          :
@@CODEL CSEG
_main :
$DGL    1 , 19
        push    hl                      ; [ INF ] 1 , 1
        subw    sp , #08H               ; [ INF ] 2 , 1
        movw    hl , sp                 ; [ INF ] 3 , 1
??bf_main :
; line   9 :    int i , prime , k , count ;
; line  10 :
; line  11 :    count = 0 ;
$DGL    0 , 4
        clrw    ax                      ; [ INF ] 1 , 1
        movw    [ hl ] , ax     ; count ; [ INF ] 1 , 1
; line  12 :
; line  13 :    for ( i = 0 ; i <= SIZE ; i++ )
$DGL    0 , 6
        movw    [ hl + 6 ] , ax ; i     ; [ INF ] 2 , 1
?L0003 :
        movw    ax , [ hl + 6 ] ; i     ; [ INF ] 2 , 1
        cmpw    ax , #0C8H      ; 200   ; [ INF ] 3 , 1
        or1     CY , a.7                ; [ INF ] 2 , 1
        skc                             ; [ INF ] 2 , 1
        bnz     $?L0004                 ; [ INF ] 2 , 4
          :

; *** Code Information ***
;
;
```

```
; $FILE C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥smp78k0r¥cc78k0r¥prime.c
;
; $FUNC main ( 8 )
;       bc = ( void )
;       CODE SIZE = 117 bytes , CLOCK_SIZE = 86 clocks , STACK_SIZE = 16 bytes
;
; $CALL printf ( 18 )
;       bc = ( pointer : ax , int : [ sp + 2 ] )
;
; $CALL putchar ( 18 )
;       bc = ( pointer : ax , int : [ sp + 2 ] )
;
; $CALL putchar( 20 )
;       bc = ( int : ax )
;
; $CALL printf ( 25 )
;       bc = ( pointer : ax , int : [ sp + 2 ] )
;
; $FUNC printf ( 31 )
;       bc = ( pointer s : ax , int i : [ sp + 4 ] )
;       CODE SIZE = 22 bytes , CLOCK_SIZE = 20 clocks , STACK_SIZE = 14 bytes
;
; $FUNC putchar ( 41 )
;       bc = ( char c : x )
;       CODE SIZE = 16 bytes , CLOCK_SIZE = 16 clocks , STACK_SIZE = 6 bytes

; Target chip : uPD78F1166_A0
; Device file : Vx.xx
```

エラー・リスト・ファイル作成指定

-e，-se

(1) -e

【記述形式】

-e[*出力ファイル名*]

- 省略時解釈

  エラー・リスト・ファイルを出力しません。

【機能】

- -e オプションは，エラー・リスト・ファイルを出力することを指示します。また，その出力先や出力ファイル名を指示します。

【用途】

- エラー・リスト・ファイルの出力先や出力ファイル名を変更したいときに，-e オプションを指定します。

【説明】

- ファイル名として，ディスク型ファイル名とデバイス型ファイル名を指定することができます。
- -e オプションを指定する際に出力ファイル名を省略すると，エラー・リスト・ファイル名は，“*入力ファイル名* .ecc”となります。
- -e オプションを指定する際に出力ファイル名の拡張子を省略すると，エラー・リスト・ファイル名は，“*出力ファイル名* .ecc”となります。
- -e オプションを指定する際にドライブ名を省略すると，カレント・ドライブにエラー・リスト・ファイルが出力されます。
- -w0 オプションが指定された場合には，ワーニング・メッセージは出力されません。

【注意】

- PM+ 使用時に，出力先を変更する場合は，[出力] タブの“エラー・リスト・ファイルの出力 (R)”の“出力パス”コンボ・ボックスに出力先を指定し，“C ソース無し [-e]”を選択してください。
- 個別コンパイラ・オプション設定時は，出力ファイル名の変更も可能です。
- [出力] タブの“出力パス”コンボ・ボックスに，ファイル名，または出力先を指定してください。

【使用例】

- エラー・リスト・ファイル prime.ecc を出力します。

```
C>cc78k0r -cf1166a0 prime.c -e
```

＜出力例＞

```
prime.c ( 18 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 20 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 26 ) : CC78K0R warning W0622 : No return value
prime.c ( 37 ) : CC78K0R warning W0622 : No return value
prime.c ( 44 ) : CC78K0R warning W0622 : No return value

Target chip : uPD78F1166_A0
Device file : Vx.xx

Compilation complete, 0 error(s) and 5 warning(s) found.
```

(2) -se

【記述形式】

-se[*出力ファイル名*]

- 省略時解釈

  エラー・リスト・ファイルを出力しません。

【機能】

- -se オプションは，エラー・リスト・ファイルに C ソース・モジュール・ファイルを付加します。また，その出力先や出力ファイル名を指示します。

【用途】

- エラー・リスト・ファイルと C ソース・モジュール・ファイルを一緒に出力したいときに，-se オプションを指定します。

【説明】

- ファイル名として，ディスク型ファイル名とデバイス型ファイル名を指定することができます。
- -se オプションを指定する際に出力ファイル名を省略すると，エラー・リスト・ファイル名は，“*入力ファイル名* .cer”となります。
- -se オプションを指定する際に出力ファイル名の拡張子を省略すると，エラー・リスト・ファイル名は，“*出力ファイル名* .cer”となります。
- -se オプションを指定する際にドライブ名を省略すると，カレント・ドライブにエラー・リスト・ファイルが出力されます。
- インクルード・ファイルに対しては，フォルダ，およびファイル名の指定を行うことはできません。
  インクルード・ファイルのファイル・タイプが“H”の場合は“her”，“C”の場合は“cer”，それ以外の場合は“er”のファイル・タイプを持つエラー・リスト・ファイルをカレント・ドライブに出力します。
- エラーがなかった場合は，C ソースは付加されません。また，その場合は，インクルード・ファイルに対しては，エラー・リスト・ファイルは作成されません。
- -w0 オプションが指定された場合には，ワーニング・メッセージは出力されません。

【注意】

- PM+ 使用時に，出力先を変更する場合は，[出力] タブの“エラー・リスト・ファイルの出力 (R)”の“出力パス”コンボ・ボックスに出力先を指定し，“C ソース付き [-se]”を選択してください。
- 個別コンパイラ・オプション設定時は，出力ファイル名の変更も可能です。
- [出力] タブの“出力ファイル”コンボ・ボックスに，ファイル名，または出力先を指定してください。

【使用例】

- エラー・リスト・ファイル prime.cer に C ソース・モジュール・ファイル prime.c を付加します。

```
C>cc78k0r -cf1166a0 prime.c -se
```

＜出力例＞

```
/*
78K0R Series C Compiler Vx.xx Error List Date : XX XXX XXXX Time : XX : XX : XX

Command         : -cf1166a0  prime.c  -se
In-file         : prime.c
Err-file        : prime.cer
Para-file       :
*/

#define TRUE    1
#define FALSE   0
#define SIZE    200

char    mark [ SIZE + 1 ] ;
main ( )
{
          :
        prime = i + i + 3 ;
        printf ( "%6d" , prime ) ;
*** CC78K0R warning W0745 : Expected function prototype
        count++ ;
        if ( ( count%8 ) == 0 ) putchar ( '\n' ) ;
*** CC78K0R warning W0745 : Expected function prototype
        for ( k = i + prime ; k <= SIZE ; k += prime )
          :
}
```

クロスリファレンス・リスト・ファイル作成指定

-x

(1) -x

【記述形式】

-x[*出力ファイル名*]

- 省略時解釈

  クロスリファレンス・リスト・ファイルを出力しません。

【機能】

- -x オプションは，クロスリファレンス・リスト・ファイルを出力することを指示します。また，その出力先や出力ファイル名を指示します。クロスリファレンス・リスト・ファイルは，シンボルの参照頻度，シンボルの定義／参照箇所を調べるのに有効です。

【用途】

- クロスリファレンス・リスト・ファイルを出力したいとき，および出力先や出力ファイル名を変更したいときに -x オプションを指定します。

【説明】

- ファイル名として，ディスク型とデバイス型ファイル名を指定することができます。
- -x オプションを指定する際に出力ファイル名を省略すると，クロスリファレンス・リスト・ファイル名は，“*入力ファイル名* .xrf”となります。
- -x オプションを指定する際に出力ファイル名の拡張子を省略すると，クロスリファレンス・リスト・ファイル名は，“*出力ファイル名* .xrf”となります。
- C0101 以外の内部エラー，F0024，E から始まる番号のコンパイル・エラーが発生した場合でも，クロスリファレンス・リスト・ファイルを作成します。ただし，ファイルの内容は保証しません。

【注意】

- PM+ 使用時に，出力先を変更する場合は，[出力] タブの“クロスリファレンス・リスト・ファイルの出力 [-x](S)”の“出力パス”コンボ・ボックスに，出力先を指定してください。
- 個別コンパイラ・オプション設定時は，出力ファイル名の変更も可能です。
- [出力] タブの“出力パス”コンボ・ボックスに，ファイル名，または出力先を指定してください。

【使用例】

- クロスリファレンス・リスト・ファイル prime.xrf を出力します。

```
C>cc78k0r -cf1166a0 prime.c -x
```

＜出力例＞

```
78K/0R Series C Compiler Vx.xx Cross reference List  Date : XX XXX XXXX Page : 1

Command   : -cf1166a0 prime.c -x
In-file   : prime.c
Xref-file : prime.xrf
Para-file :

ATTRIB MODIFY TYPE    SYMBOL          DEFINE  REFERENCE

EXTERN NEAR   array   mark               5      29      31      37
EXTERN FAR    func    printf             7      33      40
REG1          pointer s                  7      13
PARAM
REG1          int     i                  7      12
PARAM
REG1          int     j                  9      12
REG1          pointer ss                10      13
EXTERN FAR    func    putchar           16      35
REG1          char    c                 16      19
PARAM
REG1          char    d                 18      19
EXTERN FAR    func    main              22
REG1          int     i                 24      28      28      28      29
    30      30      30      31      32      32
                                          36
REG1          int     prime             24      32      33      36      36
REG1          int     k                 24      36      36      36      37
REG1          int     count             24      26      34      35      40
              #define TRUE               1      29
              #define FALSE              2      37
              #define SIZE               3       5      28      30      36


 Target chip : uPD78F1166_A0
 Device file : Vx.xx
```

リスト形式指定

-lw，-ll，-lt，-lf，-li

(1) -lw

【記述形式】

```
-lw[ 文字数 ]
```

- 省略時解釈

  -lw132（コンソール出力の場合は 80 文字とします）

【機能】

- -lw オプションは，各種リスト・ファイルの 1 行の文字数を指示します。

【用途】

- 各種リスト・ファイルの 1 行の文字数を変更したいときに，-lw オプションを指定します。

【説明】

- -lw オプションで指定可能な文字数の範囲は，ターミネータ（CR，LF）は含めないで次のとおりです。

  72 ≦ 1 行に印字する文字数 ≦ 132

- 文字数を省略した場合は，1 行の文字数は 132 文字となります（コンソール出力の場合には，80 文字までとなります）。
- リスト・ファイルが何も指定されない場合，-lw オプションは無効となります。

【使用例】

- クロスリファレンス・リスト・ファイル prime.xrf の 1 行の文字数を 72 文字とします。

```
C>cc78k0r -cf1166a0 prime.c -x -lw72
```

(2) -ll

【記述形式】

```
-ll[ 行数 ]
```

- 省略時解釈

  改ページしません。

【機能】

- -ll オプションは，各種リスト・ファイルの 1 ページの行数を指示します。

【用途】

- 各種リスト・ファイルの 1 ページの行数を変更したいときに，-ll オプションを指定します。

【説明】

- -ll オプションで指定可能な行数の範囲は次のとおりです。

  20 ≦ 1 ページに印字する行数 ≦ 65535

- -ll0 を指定すると，改ページしません。
- 行数を省略した場合は，改ページしません。
- リスト・ファイルが何も指定されない場合，-ll オプションは無効となります。

【使用例】

- クロスリファレンス・リスト・ファイル prime.xrf の 1 ページの行数を 20 行とします。

```
C>cc78k0r -cf1166a0 prime.c -x -ll20
```

(3) -lt

【記述形式】

```
-lt[ 文字数 ]
```

- 省略時解釈

  -lt8

【機能】

- -lt オプションは，ソース・モジュール中の HT（Horizontal Tabulation）コードを，各種リスト上でいくつかのブランク（空白）に置き換えて出力する（タビュレーション処理）ための，基本となる文字数を指示します。

【用途】

- -lw オプションで各種リストの 1 行の文字数を少なく指定した場合などに HT コードによるブランクを少なくし，文字数を節約するために -lt オプションを指定します。

【説明】

- -lt オプションで指定可能な文字数の範囲は次のとおりです。

  0 ≦ 指定可能な文字数 ≦ 8

- -lt0 を指定した場合，タビュレーション処理は行わず，タブ・コードを出力します。
- 文字数を省略した場合は，タブの展開文字数は 8 文字となります。
- リスト・ファイルが何も指定されない場合，-lt オプションは無効となります。

【使用例】

- -lt オプションの省略により，-lt8 オプションが指定されたものと見なされ，HT コードによるブランクを 8 とします。

```
C>cc78k0r -cf1166a0 prime.c -p
```

- HT コードによるブランクを 1 とします。

```
C>cc78k0r -cf1166a0 prime.c -p -lt1
```

(4) -lf

【記述形式】

```
-lf
```

- 省略時解釈

　改ページ・コードを付加しません。

【機能】

- -lf オプションは，各種リスト・ファイルの最後に改ページ・コードを付加することを指示します。

【説明】

- リスト・ファイルが何も指定されない場合，-lf オプションは無効となります。

【使用例】

- アセンブラ・ソース・モジュール・ファイル prime.asm の最後に改ページ・コードを付加します。

```
C>cc78k0r -cf1166a0 prime.c -a -lf
```

(5) -li

【記述形式】

```
-li
```

- 省略時解釈

  インクルード・ファイルの C ソースを付加しません。

【機能】

- -li オプションは，C ソース・コメント付きアセンブラ・ソース・モジュール・ファイルに，インクルード・ファイルの C ソースも付加します。

【説明】

- -sa オプションを指定しない場合は，このオプションは無視されます。

【使用例】

- C ソース・コメント付きアセンブラ・ソース・モジュール・ファイル prime.asm に，インクルード・ファイルの C ソースも付加します。

```
C>cc78k0r -cf1166a0 prime.c -sa -li
```

ワーニング出力指定

-w

(1) -w

【記述形式】

```
-w[ レベル ]
```

- 省略時解釈

  -w1

【機能】

- -w オプションは，ワーニング・メッセージをコンソールに出力するか否かを指定します。

【用途】

- ワーニング・メッセージをコンソールに出力する／しないを指定します。
  また，詳細なメッセージを出力させることも可能です。

【説明】

- ワーニング・メッセージのレベルには，次のものがあります。

レベル	説明
0	ワーニング・メッセージを出力しません。
1	通常のワーニング・メッセージを出力します。
2	詳細なワーニング・メッセージを出力します。

- -e，-se オプションが指定された場合には，エラー・リスト・ファイルにもワーニング・メッセージが出力されます。
- レベル 0 を指定すると，ワーニング・メッセージをコンソール，エラー・リスト・ファイル（-e，-se 指定時）に出力しません。

【使用例】

- -w オプションの省略により，-w1 オプションが指定されたものと見なされ，通常のワーニング・メッセージを出力します。

```
C>cc78k0r -cf1166a0 prime.c
```

実行状態表示指定

-v/-nv

(1) -v/-nv

【記述形式】

```
-v
-nv
```

- 省略時解釈

    -nv

【機能】

- -v オプションは，現在のコンパイルの実行状態をコンソールに出力します。
- -nv オプションは，-v オプションを無効にします。

【用途】

- コンパイルの実行状況を確認したいときに指定します。

【説明】

- フェーズ名，および処理中の関数名を出力します。
- -v と -nv の両オプションが同時に指定された場合は，後ろに指定したものが優先となります。

【使用例】

- 現在のコンパイルの実行状態をコンソールに出力します。

```
C>cc78k0r -cf1166a0 prime.c -v
```

パラメータ・ファイル指定

-f

(1) -f

【記述形式】

```
-f ファイル名
```

- 省略時解釈

  コマンド行上からのみオプション，入力ファイル名の入力が可能

【機能】

- -f オプションは，オプション，あるいは入力ファイル名を指定のファイルから入力することを指示します。

【用途】

- コンパイル時に複数のオプションを入力するため，コマンド行では CC78K0R の起動に必要な情報を指定しきれないときに -f オプションを指定します。
- 繰り返し同じようにオプションを指定しコンパイルする際には，それらをパラメータ・ファイルに記述しておき，-f オプションを指定します。

【説明】

- パラメータ・ファイルのネストは許されません。
- パラメータ・ファイル中に記述可能な文字数に，制限はありません。
- 空白とタブをオプション，あるいは入力ファイル名の区切りとします。
- パラメータ・ファイル中に記述したオプション，あるいは入力ファイル名は，コマンド行上のパラメータ・ファイル指定のあった位置に展開されます。
- 展開されたオプションの優先順位は，後ろに指定したものが優先となります。
- “;”，および“#”以降に記述された文字は，行末まですべてコメントと解釈します。

【使用例】

- パラメータ・ファイル prime.pcc の内容

```
; parameter file
prime.c -cf1166a0 -aprime.asm -e -x
```

パラメータ・ファイル prime.pcc を使用してコンパイルします。

```
C>cc78k0r -fprime.pcc
```

テンポラリ・ファイル作成フォルダ指定

-t

(1) -t

【記述形式】

```
-t フォルダ
```

- 省略時解釈

  環境変数 TMP で指定したドライブ・フォルダに，テンポラリ・ファイルを作成します。環境変数 TMP による指定がない場合は，カレント・ドライブ，カレント・フォルダに，テンポラリ・ファイルを作成します。

【機能】

- -t オプションは，テンポラリ・ファイルを作成するドライブ，フォルダを指示します。

【用途】

- テンポラリ・ファイルの作成場所を指定することができます。

【説明】

- 以前に作成されたテンポラリ・ファイルが存在している場合でも，ファイル保護がされていなければ，次の作成時には上書きします。
- テンポラリ・ファイルは，必要とするメモリ・サイズがある場合は，メモリに展開します。
  必要とするメモリ・サイズがなくなった場合は，メモリの内容を指定されたフォルダ下にテンポラリ・ファイルを作成してファイルに書き出し，それ以降のテンポラリ・ファイルに対するアクセスは，メモリ上でなく，ファイルに対して行います。
- テンポラリ・ファイルは，コンパイル終了時に削除されます。また，[CTRL] キー+[C] キーを押すことによってコンパイルが中止されたときも，削除されます。

【使用例】

- テンポラリ・ファイルをフォルダ tmp に作成します。

```
C>cc78k0r -cf1166a0 prime.c -ttmp
```

ヘルプ指定

--/-?/-h

(1) --/-?/-h

【記述形式】

```
--/-?/-h
```

- 省略時解釈

  表示しません。

【機能】

- --/-?/-h オプションは，オプションの簡単な説明，デフォルト・オプションなどのヘルプ・メッセージをコンソールに表示します（コマンド・ラインのみ有効[注]）。

  注 PM+ 上では，このオプションは指定しないでください。PM+ 上でヘルプを参照する場合は，[コンパイラオプションの設定] ダイアログで [ヘルプ] ボタンを押してください。

【用途】

- オプションとその説明が表示されます。CC78K0R 実行時に参照してください。

【説明】

- --/-?/-h オプションを指定すると，他のコンパイラ・オプションはすべて無効となります。
- ヘルプ・メッセージの続きを参照する場合は [Enter] キーを，表示を途中で終了する場合には，[Enter] キー以外の文字を入力したあとに [Enter] キーを入力してください。

【使用例】

- ヘルプ・メッセージをコンソールに表示します。

```
C>cc78k0r -h
```

機能拡張指定

-z/-nz

(1) -z/-nz

【記述形式】

```
-z 種別（複数指定可能）
-nz
```

- 省略時解釈

  -nz

【機能】

- -z オプションは，拡張機能を有効とします。
- -nz オプションは，-z オプションを無効とします。
- 種別を省略することはできません。省略した場合は，致命的エラー（F0012）となります。

【用途】

- 78K0R シリーズ拡張機能に対し，**【説明】**に示す機能を持たせます。

【説明】

- -z オプションの種別指定には，次のものがあります。

種別指定	説明
p	“//”以降改行までをコメントと解釈します。
c	“/* */”コメントのネストを許します。
s 注	コメント中の漢字種別を SJIS コードと解釈します。
e 注	コメント中の漢字種別を EUC コードと解釈します。
n 注	コメント中に漢字コードが無いと解釈します。
b	char/unsigned char 型引数／返り値を int 拡張しません。
a	ANSI 規定外の機能を無効とし，ANSI 規定の一部の機能を有効とします。 具体的には次の動作を行います。 - 次のものは予約語ではなくなります。 callt/norec/sreg/bit/boolean/#asm/#endasm - トライグラフ・シーケンス（3 文字表記）が有効となります。 - コンパイラ定義マクロ __STDC__ は 1 となります。 - far ポインタの関係演算を 3 バイトで行うことで，データを 64K バイト境界の最後の 1 バイトに配置可能とします。 - char 型ビット・フィールドに対し，次のワーニングを出力します。 （CC78K0R warning W0787 : Bit field type is char） - -qc，-zp，-zc オプションに対し，-w2 指定時，次のワーニングを出力します。 （CC78K0R warning W0029 : ' -QC ' option is not portable） （CC78K0R warning W0031 : ' -ZP ' option is not portable） （CC78K0R warning W0032 : ' -ZC ' option is not portable） - 各種 #pragma 文に対し，-w2 指定時，次のワーニングを出力します。 （CC78K0R warning W0849 : #pragma statement is not portable） - __asm 文に対し，-w2 指定時，次のワーニングを出力し，アセンブル出力は行われます。 （CC78K0R warning W0850 : Asm statement is not portable） - #asm ～ #endasm ブロックに対し，-w2 指定時，次のエラーなどを出力します。 （CC78K0R error E0801 : Undefined control など）
f	フラッシュ用オブジェクトを出力します。

注 s，e，n は，同時に指定することはできません。

【使用例】

- C ソース prime.c 中の“//”以降改行までをコメントと解釈します。また，“/* */”コメントのネストを許します。

```
C>cc78k0r -cf1166a0 prime.c -zpc
```

デバイス・ファイルのサーチ・パス

-y

(1) -y

【記述形式】

-y *フォルダ*

- 省略時解釈

  通常のサーチ・パスのみ

  備考 通常のサーチ・パスは，次のとおりです。

  (1) < ..¥..¥..¥dev >（cc78k0r.exe の起動されたパスに対して）

  (2) CC78K0R の起動されたパス

  (3) カレント・フォルダ

  (4) 環境変数 PATH

【機能】

- -y オプションは，デバイス・ファイル読み込みのためのサーチ・パスとして，指定されたパスを最初に探します。存在しなければ，通常のパスから探します。

【用途】

- デバイス・ファイルを通常のサーチ・パスにはない特定のフォルダにインストールする場合，そのパスをこのオプションで指定します。

【注意】

- PM+ 使用時は，[プロジェクトの設定] ダイアログの“デバイス名 (D)”でデバイス・ファイルを登録する際に，フォルダが決定されます。したがって，コンパイラのオプション設定ではオプションを指定する必要はありません。

【使用例】

- デバイス・ファイル読み込みのため，“A:¥tmp¥dev”を最初にサーチします。

```
C>cc78k0r -cf1166a0 -yA:¥tmp¥dev
```

メモリ・モデル指定

-m

(1) -m

【記述形式】

-m *種別*

- 省略時解釈

  -mm

【機能】

- -m オプションは，コンパイル時のメモリ・モデルの種類を指定します。
- 種別を同時に複数指定することはできません。
- 種別を省略することはできません。省略した場合は，致命的エラー（F0012）となります。

【用途】

- メモリ・モデルを指定することにより，関数，および変数の near/far 領域を指定します。
- C ソース上で関数，および変数に __near/__far 修飾子を記述した場合は，__near/__far 修飾子による near/far 領域指定が優先されます。

【説明】

- -m オプションの種別指定には，次のものがあります。

種別指定	メモリ・モデル	説明
s	スモール・モデル	コード部最大 64K バイト，データ部最大 64K バイト，合計して 128K バイトとみなして，near/far 領域を指定します。
m	ミディアム・モデル	コード部最大 1M バイト，データ部最大 64K バイト，合計して 1M バイトとみなして，near/far 領域を指定します。
c	コンパクト・モデル	コード部最大 64K バイト，データ部最大 1M バイト，合計して 1M バイトとみなして，near/far 領域を指定します。
l	ラージ・モデル	コード部最大 1M バイト，データ部最大 1M バイト，合計して 1M バイトとみなして，near/far 領域を指定します。

注意 データ部，またはコード部が最大 64K バイトのメモリ・モデルを指定しても，__far 修飾子がある関数，および変数は，最大 1M バイトの空間に配置することができます。

メモリ・モデルの指定は，__near/__far 修飾子を持たない関数，および変数の配置を指定するものです。

【使用例】

- コンパイル時のメモリ・モデルは，スモール・モデルとなります。

```
C>cc78k0r -cf1166a0 prime.c -ms
```

第 6 章 C コンパイラの出力ファイル

この章では，CC78K0R が出力するファイルについて説明します。

CC78K0R は，次のファイルを出力します。

- オブジェクト・モジュール・ファイル
- アセンブラ・ソース・モジュール・ファイル
- エラー・リスト・ファイル
- プリプロセス・リスト・ファイル
- クロスリファレンス・リスト・ファイル

6.1 オブジェクト・モジュール・ファイル

オブジェクト・モジュール・ファイルは，C ソース・プログラムのコンパイル結果のバイナリ・イメージ・ファイルです。

また，デバッグ情報出力指定オプション -g により，デバッグ情報を含めることができます。

6.2 アセンブラ・ソース・モジュール・ファイル

アセンブラ・ソース・モジュール・ファイルは，C ソース・プログラムのコンパイル結果の ASCII イメージのリストで，C ソース・プログラムに対応したアセンブリ言語のソース・モジュール・ファイルです。

また，アセンブラ・ソース・モジュール・ファイル作成指定オプション -sa により，コメントとして C ソース・プログラムを含めることができます。

【出力形式】

```
    ; 78K0R Series C Compiler V(1)x.xx Assembler Source
    ;                                       Date: (2)xxxxx  Time: (3)xxxxx

    ; Command       : (4)-cf1166a0 prime.c –sa
    ; In-file       : (5)prime.c
    ; Asm-file      : (6)prime.asm
    ; Para-file     : (7)

    $PROCESSOR ((8) f1166a0 )
(9) $DEBUG
(10)$NODEBUGA
(11)$KANJICODE SJIS
(12)$TOL_INF 03FH , 100H , 00H , 00H , 00H

(13)$DGS    FIL_NAM , .file ,       034H , 0FFFEH , 03FH , 067H , 01H , 00H
                :
(14)        EXTRN   _@RTARG0
                :
    ; line (15)1    : (16)#define   TRUE    1
    ; line (15)2    : (16)#define   FALSE   0
    ; line (15)3    : (16)#define   SIZE    200
                :
(14)_main :
(17)$DGL    1 , 14
(14)        push    hl                          ; (21) [ INF ] 1 , 1
(14)        subw    sp , #08H                   ; (21) [ INF ] 2 , 1
(14)        movw    ax , sp                     ; (21) [ INF ] 2 , 1
(14)        movw    hl , ax                     ; (21) [ INF ] 1 , 1
                :
(18)??bf_main :
                :
    ; (22)*** Code Information ***
    ;
    ; (23)  $FILE   C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥Vx.xx¥
smp78k0r¥CC78K0R¥prime.c
    ;
    ; (24)  $FUNC   main ( 8 )
    ; (25)          bc = ( void )
    ; (26)         CODE SIZE = 116 bytes , CLOCK_SIZE = 86 clocks , STACK_SIZE =
16 bytes
    ;
    ; (27)  $CALL   printf ( 18 )
    ; (28)          bc = ( pointer:ax , int : [ sp + 2 ] )
    ;
    ; (27)  $CALL   putchar ( 20 )
    ; (28)          bc = ( int : ax ) ;
    ;
```

```
    ; (27)  $CALL   printf ( 25 )
    ; (28)          bc = ( pointer : ax , int : [ sp + 2 ] )
    ;
    ; (24)  $FUNC   printf ( 31 )
    ; (25)          bc = ( pointer s :ax , int i : [ sp + 4 ] )
    ; (26)          CODE SIZE = 23 bytes , CLOCK_SIZE = 22 clocks , STACK_SIZE =
14 bytes
    ;
    ; (24)  $FUNC   putchar ( 41 )
    ; (25)          bc = ( char c : x )
    ; (26)          CODE SIZE = 16 bytes , CLOCK_SIZE = 18 clocks , STACK_SIZE =
6 bytes

    ; Target chip   : (19)uPD78F1166_A0
    ; Device file   : (20)Vx.xx
```

項番	内容	桁数	形式
(1)	バージョン番号	4（固定）	“*x.yz*”の形式で表します。
(2)	日付	11（固定）	システム日付。 “*DD Mmm YYYY*”の形式で表します。
(3)	時間	8（固定）	システム時間。 “*HH* : *MM* : *SS*”の形式で表します。
(4)	コマンド行	―	コマンド行の“CC78K0R”以降を出力します。80 カラム目からは，次行の 15 カラム目から出力します。ただし，1 カラム目に“ ; ”を出力します。1 個以上の空白タブは，1 個の空白で置き換えます。
(5)	C ソース・モジュール・ファイル名	OS で許可している文字数	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.c”を付加します。80 カラム目からは，次行の 15 カラム目から出力します。ただし，1 カラム目に“ ; ”を出力します。
(6)	アセンブラ・ソース・モジュール・ファイル名	OS で許可している文字数	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.asm”を付加します。80 カラム目からは，次行の 15 カラム目から出力します。ただし，1 カラム目に“ ; ”を出力します。
(7)	パラメータ・ファイルの内容	―	パラメータ・ファイルの内容を出力します。 80 カラム目からは，次行の 15 カラム目から出力します。ただし，1 カラム目に“ ; ”を出力します。1 個以上の空白タブは，1 個の空白で置き換えます。
(8)	デバイス種別	最大 6（可変）	-c オプションで指定された文字列です。
(9)	デバッグ情報	最大 8（可変）	DEBUG コントロールを出力します。 $DEBUG，あるいは $NODEBUG のいずれかです。
(10)	アセンブラのデバッグ情報の制御	9（固定）	NODEBUGA コントロールを出力します。 $NODEBUGA を出力します。

項番	内容	桁数	形式
(11)	漢字種別情報	最大 15（可変）	漢字種別を出力します。 $KANJICODE SJIS，$KANJICODE EUC，あるいは $KANJICODE NONE を出力します。
(12)	ツール情報	37（固定）	ツール情報，バージョン番号，エラー情報，オプションの有無などを出力します（$TOL_INF で始まる情報）。
(13)	シンボル情報	−	シンボル情報を出力します（$DGS で始まる情報）。 デバッグ情報出力指定オプションを指定した場合にのみ出力します。ただし，-g1 が指定された場合は出力しません。
(14)	アセンブラ・ソース本体	−	コンパイル結果のアセンブラ・ソースを出力します。
(15)	行番号	4（固定）	C ソース・モジュール・ファイルの行番号（右詰めゼロ・サプレスの 10 進数）です。
(16)	C ソース	−	入力 C ソース・イメージです。 80 カラム目からは，次行の 16 カラム目から出力します。ただし，1 カラム目に“ ; ”を出力します。
(17)	ライン・ナンバ情報	−	ライン・ナンバ・エントリ用の行番号（$DGL で始まる情報）です。 デバッグ情報出力指定オプションを指定した場合のみ出力します。ただし，-g1 が指定された場合は出力しません。
(18)	シンボル情報作成用ラベル	最大 34（可変）	関数のラベル情報です（?? で始まる情報）。 デバッグ情報出力指定オプションを指定した場合にのみ出力します。
(19)	コンパイルの対象品種名	最大 15（可変）	コマンド行オプション -c，またはソース・ファイルにおいて指定された対象デバイス名を表示します。
(20)	デバイス・ファイル・バージョン	6（固定）	入力したデバイス・ファイルのバージョン番号を表示します。
(21)	サイズ，クロック	−	出力命令に対する，サイズ，クロック数を出力します（; [INF] で始まる情報）。 クロック数は，内部 RAM 領域，SFR 領域をアクセスした場合，またはデータ・アクセスをしない場合のクロック数を出力します。 また，条件分岐命令の場合，条件成立時のクロック数を出力します。 ハザードは，考慮していません。
(22)	関数情報（開始）	−	関数情報の開始を示します。
(23)	関数情報（ファイル名）	−	対象ソース・ファイル名をフルパスで出力します（; $FILE で始まる情報）。
(24)	関数情報（定義関数）	−	関数名，および定義行番号を 10 進で出力します（; $FUNC で始まる情報）。
(25)	関数情報（定義関数の返り値，引数）	−	定義関数の返り値レジスタ，引数情報（レジスタ，またはスタック位置）を出力します。

項番	内容	桁数	形式
(26)	関数情報（定義関数のサイズ，クロック，スタック）	－	定義関数に対する静的に計算したサイズ，クロック，最大消費スタックを出力します。 ここで表示されるスタック量は，関数本体で使用するスタック量のみです。 例えば，関数内で別の関数を呼び出している場合，呼び出し先の関数で使用したスタック分は，呼び出し元の関数のスタック量には加算されません。 また，CLOCK_SIZE は項番 (21) のクロックを加算したものです。
(27)	関数情報（呼び出し関数）	－	関数名と関数呼び出し行番号を 10 進で出力します（; $CALL で始まる情報）。
(28)	関数情報（呼び出し関数の返り値，引数）	－	関数呼び出し時の，返り値レジスタと引数情報（レジスタ，またはスタック位置）を出力します。

6.3 エラー・リスト・ファイル

エラー・リスト・ファイルは，コンパイル中に発生したワーニングやエラー・メッセージの集まりでできています。

コンパイラ・オプションを指定することにより，エラー・リスト中に C ソース・プログラムを付加することができます。C ソース・プログラムを含むエラー・リスト・ファイルは，C ソース・プログラムを修正し，リスト・ヘッダなどのコメントを削除することにより，C ソース・モジュール・ファイルとして使用することができます。

6.3.1 C ソース付きのエラー・リスト・ファイル

【出力形式】

```
/*
78K0R Series C Compiler V(1)x.xx Error List  Date: (2)xxxxx  Time: (3)xxxxx

Command         : (4)-cf1166a0 prime.c -se
C-file          : (5)prime.c
Err-file        : (6)prime.cer
Para-file       : (7)
*/

(8)#define      TRUE    1
(8)#define      FALSE   0
(8)#define      SIZE    200

(8)char         mark [ SIZE + 1 ] ;

(8)main ( )
(8){
(8)     int     i , prime , k , count ;
(8)     cont = 0 ;
  *** CC78K0R error(9)E0711 :(10)Undeclared 'cont' ; function 'main'
(8)     for ( i = 0 ; i <= SIZE ; i++ )
(8)             mark [ i ] = TRUE ;
(8)     for ( i = 0 ; i <= SIZE ; i++ ) {
(8)             if ( mark [ i ] ) {
                        prime = i + i + 3 ;
                        printf ( "%6d" , prime ) ;
  *** CC78K0R warning(9)W0745 :(10)Expected function prototype
          :
/*
(11)Target chip : uPD78F1166_A0
(12)Device file : Vx.xx
Compilation complete, (13)1 error(s) and (14)5 warning(s) found.
*/
```

項番	内容	桁数	形式
(1)	バージョン番号	4（固定）	“*x.yz*”の形式で表します。
(2)	日付	11（固定）	システム日付。 “*DD Mmm YYYY*”の形式で表します。
(3)	時間	8（固定）	システム時間。 “*HH* : *MM* : *SS*”の形式で表します。
(4)	コマンド行	－	コマンド行の“CC78K0R”以降を出力します。80 カラム目からは，次行の 13 カラム目から出力します。1 個以上の空白タブは，1 個の空白で置き換えます。
(5)	C ソース・モジュール・ファイル名	OS で許可している文字数（可変）	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.c”を付加します。80 カラム目からは，次行の 13 カラム目から出力します。
(6)	エラー・リスト・ファイル名	OS で許可している文字数（可変）	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.cer”を付加します。80 カラム目からは，次行の 13 カラム目から出力します。
(7)	パラメータ・ファイルの内容	－	パラメータ・ファイルの内容を出力します。 80 カラム目からは，次行の 13 カラム目から出力します。1 個以上の空白タブは，1 個の空白で置き換えます。
(8)	C ソース	－	入力 C ソース・イメージです。 80 カラム目以降も折り返しを行わず出力します。
(9)	エラー・メッセージ番号	5（固定）	エラー番号を“*#nnnn*”の形式で出力します。 # は，アボート・エラーの場合は F，フェイタル・エラーの場合は E，内部エラーの場合は C，ワーニング・エラーの場合は W となります。 *nnnn* はエラー番号で，10 進数 4 桁で表します（ゼロ・サプレスしません）。
(10)	エラー・メッセージ	－	「第 9 章 エラー・メッセージ」を参照してください。80 カラム以降も折り返しを行わず出力します。
(11)	コンパイルの対象品種名	最大 15（可変）	コマンド行オプション -c，またはソース・ファイルにおいて指定された対象デバイス名を表示します。
(12)	デバイス・ファイル・バージョン	6（固定）	入力したデバイス・ファイルのバージョン番号を表示します。
(13)	エラー個数	4（固定）	右詰めゼロ・サプレスの 10 進数。
(14)	ワーニング個数	4（固定）	右詰めゼロ・サプレスの 10 進数。

6.3.2 エラー・メッセージのみのエラー・リスト・ファイル

【出力形式】

```
(1)prime.c ( (2)18 ) : CC78K0R warning (3)W0745 : (4)Expected function prototype
(1)prime.c ( (2)20 ) : CC78K0R warning (3)W0745 : (4)Expected function prototype
(1)prime.c ( (2)26 ) : CC78K0R warning (3)W0622 : (4)No return value
(1)prime.c ( (2)37 ) : CC78K0R warning (3)W0622 : (4)No return value
(1)prime.c ( (2)44 ) : CC78K0R warning (3)W0622 : (4)No return value

Target chip : (7)uPD78F1166_A0
Device file : (8)Vx.xx

Compilation complete, (5)0 error(s) and (6)5 warning(s) found
```

項番	内容	桁数	形式
(1)	C ソース・モジュール・ファイル名	OS で許可している文字数	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.c”を付加します。
(2)	行番号	5（固定）	右詰めゼロ・サプレスの 10 進数。
(3)	エラー・メッセージ番号	5（固定）	エラー番号を“*#nnnn*”の形式で出力します。 # は，アボート・エラーの場合は F，フェイタル・エラーの場合は E，内部エラーの場合は C，ワーニング・エラーの場合は W となります。 *nnnn* はエラー番号で，10 進数 4 桁で表します（ゼロ・サプレスしません）。
(4)	エラー・メッセージ	－	「第 9 章 エラー・メッセージ」を参照してください。
(5)	エラー個数	4（固定）	右詰めゼロ・サプレスの 10 進数。
(6)	ワーニング個数	4（固定）	右詰めゼロ・サプレスの 10 進数。
(7)	コンパイルの対象品種名	最大 15（可変）	コマンド行オプション，またはソース・ファイル中において指定された対象品種名を表示します。
(8)	デバイス・ファイル・バージョン	6（固定）	入力したデバイス・ファイルのバージョン番号を表示します。

6.4 プリプロセス・リスト・ファイル

プリプロセス・リスト・ファイルは，C ソース・プログラムのプリプロセス処理のみを行った結果の ASCII イメージ・ファイルです。

-k オプションを指定する際，処理種別として“n”を指定しなければ，C ソース・モジュール・ファイルとして使用することができます。-kd を指定すると，#define の展開を行ったリストを出力します。

【出力形式】

＜ PAGEWIDTH = 80 の場合＞

```
/*
78K0R Series C Compiler V(1)x.xx Preprocess List  Date :(2)xxxxx Page :(3)xxx

Command         :(4)-cf1166a0 prime.c -p -lw80
In-file         :(5)prime.c
PPL-file        :(6)prime.ppl
Para-file       :(7)
*/

    (8)1 : (9)#define  TRUE    1
    (8)2 : (9)#define  FALSE   0
    (8)3 : (9)#define  SIZE    200
    (8)4 : (9)
    (8)5 : (9)char     mark [ SIZE + 1 ] ;
    (8)6 : (9)
/*
    (10)Target chip : uPD78F1166_A0
    (11)Device file : Vx.xx
*/
```

項番	内容	桁数	形式
(1)	バージョン番号	4（固定）	“*x.yz*”の形式で表します。
(2)	日付	11（固定）	システム日付。 “*DD Mmm YYYY*”の形式で表します。
(3)	ページ数	4（固定）	右詰めゼロ・サプレスの 10 進数。
(4)	コマンド行	－	コマンド行の“CC78K0R”以降を出力します。1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。1 個以上の空白タブは，1 個の空白で置き換えます。
(5)	C ソース・モジュール・ファイル名	OS で許可している文字数	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.c”を付加します。1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。
(6)	プリプロセス・リスト・ファイル名	OS で許可している文字数	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.ppl”を付加します。1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。

項番	内容	桁数	形式
(7)	パラメータ・ファイルの内容	－	パラメータ・ファイルの内容を出力します。 1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。ただし，1 カラム目に “;” を出力します。1 個以上の空白タブは，1 個の空白で置き換えます。
(8)	行番号	5（固定）	右詰めゼロ・サプレスの 10 進数。
(9)	C ソース	－	入力 C ソースです。 1 行に入らない場合は，超過分を次行の 9 カラム目から出力します。
(10)	コンパイルの対象品種名	最大 15（可変）	コマンド行オプション，またはソース・ファイルにおいて指定された対象品種名を表示します。
(11)	デバイス・ファイル・バージョン	6（固定）	入力したデバイス・ファイルのバージョン番号を表示します。

6.5 クロスリファレンス・リスト・ファイル

クロスリファレンス・リスト・ファイルは，C ソース・プログラム中で宣言，定義，参照されている関数名，変数名などの識別子の一覧表です。属性やその行番号などの情報も含んでいます。これらを出現順に出力します。

【出力形式】

＜ PAGEWIDTH = 80 の場合＞

```
78K0R Series C Compiler V(1)x.xx Cross reference List
                                                    Date: (2)xxxxx Page: (3)xxx

Command         : (4) -cf1166a0 prime.c -x -lw80
In-file         : (5)prime.c
Xref-file       : (6)prime.xrf
Para-file       : (7)
Inc-file        : [ n ] (8)

(9)ATTRIB   (10)MODIFY  (11)TYPE    (12)SYMBOL  (13)DEFINE  (14)REFERENCE

EXTERN      NEAR        array       mark        5           14  16  22
EXTERN      FAR         func        main        7
AUTO1                   int         i           9           13  13  13  14
                                                            15  15  15  16
                                                            17  17  21
AUTO1                   int         prime       9           17  18  21  21
AUTO1                   int         k           9           21  21  21  22
AUTO1                   int         count       9           11  19  20  25
  :
/*(15)Target chip : uPD78F1166_A0
  (16)Device file : Vx.xx */
```

項番	内容	桁数	形式
(1)	バージョン番号	4	“*x.yz*”の形式で表します。
(2)	日付	11（固定）	システム日付。 “*DD Mmm YYYY*”の形式で表します。
(3)	ページ数	4（固定）	右詰めゼロ・サプレスの 10 進数。
(4)	コマンド行	ー	コマンド行の“CC78K0R”以降を出力します。1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。1 個以上の空白タブは，1 個の空白で置き換えます。
(5)	C ソース・モジュール・ファイル名	OS で許可している文字数	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.c”を付加します。1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。
(6)	クロスリファレンス・リスト・ファイル名	OS で許可している文字数	指定されたファイル名を出力します。 ファイル・タイプが省略されている場合は，“.xrf”を付加します。1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。

項番	内容	桁数	形式
(7)	パラメータ・ファイルの内容	―	パラメータ・ファイルの内容を出力します。 1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。1 個以上の空白タブは，1 個の空白で置き換えます。
(8)	インクルード・ファイル	OS で許可している文字数	C ソース中で指定されたファイル名を出力します。 n は 1 で始まる数字でインクルード・ファイル番号を示します。1 行に入らない場合は，超過分を次行の 13 カラム目から出力します。インクルード・ファイルがない場合は，この行は出力されません。
(9)	シンボル属性	6（固定）	シンボル属性を表します。 外部変数は EXTERN，static 変数は外部が EXSTC で内部が INSTC，auto 変数は AUTOnn，レジスタ変数は REGnn（nn は 1 で始まる数字でスコープを示します），typedef 宣言は外部が EXTYP で内部が INTYP，ラベルは LABEL，構造体・共用体のタグは TAG，メンバは MEMBER，関数のパラメータは PARAM です。
(10)	シンボル修飾属性	6（固定）	シンボル修飾属性を表します。 左詰めです。 CONST（const 変数），VLT（volatile 変数），CALLT（callt 関数），NOREC（norec 関数），SREG（sreg・bit 変数），RWSFR（sfr 変数），ROSFR（リード・オンリーの sfr 変数），WOSFR（ライト・オンリーの sfr 変数），VECT（割り込み関数），NEAR（near 領域配置の変数，関数），FAR（far 領域配置の変数，関数）の属性があります。
(11)	シンボル・タイプ	7（固定）	シンボルのタイプを表します。 char，int，short，long，field となります。 unsigned タイプは，それぞれ先頭に u が付加されます。 他に void，float，double，ldouble（longdouble），func，array，pointer，struct，union，enum，bit，inter，#define があります。
(12)	シンボル名	15（固定）	15 文字を越えた場合は，1 行に収まるときはシンボル名をそのまま出力し，1 行に収まらないときは超過分を次行の 23 カラム目から出力し，(13)，(14) の項目を次行の 39 カラム目から出力します。
(13)	シンボル定義行番号	7（固定）	シンボルの定義された行番号とファイル名で，行番号（5 桁）：インクルード・ファイル番号の形式で表します。
(14)	シンボル参照行番号	7（固定）	シンボルを参照している行番号とファイル名で行番号（5 桁）：インクルード・ファイル番号の形式で表します。1 行に入らないときは，超過分を次行の 47 カラム目から出力します。
(15)	コンパイルの対象品種名	最大 15（可変）	コマンド行オプション，またはソース・ファイルにおいて指定された対象品種名を表示します。

項番	内容	桁数	形式
(16)	デバイス・ファイル・バージョン	6（固定）	入力したデバイス・ファイルのバージョン番号を表示します。

第 7 章 C コンパイラの活用法

この章では，CC78K0R を効率よく活用すための手法を紹介します。

7.1 効率良く作業する（EXIT ステータス機能）

CC78K0R は，コンパイル終了時にコンパイル中に発生した最大のエラー・レベルを EXIT ステータスとして，OS に返します。

EXIT ステータスを次に示します。

表 7-1 EXIT ステータス一覧

状況	EXIT ステータス
正常終了時	0
WARNING あり	0
FATAL ERROR あり	1
ABORT 時	2

PM+ を利用せず，コマンド行で CC78K0R を起動する場合，これらをバッチ・ファイルで利用することにより効率良く作業を進めることができます。

【使用例】

```
cc78k0r -cf1166a0 %1
IF ERRORLEVEL 1 GOTO ERR
cc78k0r -cf1166a0 %2
IF ERRORLEVEL 1 GOTO ERR
GOTO EXIT
ERR
echo Some error found.
EXIT
```

【説明】

- %1 に渡された C ソースをコンパイルした時点で，FATAL ERROR が生じたとします。本来ならば，エラー・メッセージを出力したあと処理を続行しますが，EXIT ステータスに 1 が返されることを利用して，次の %2 の C ソースの処理を行わずに実行を停止することができます。

7.2 開発環境を整える（環境変数）

CC78K0R は，次の環境変数をサポートしています。

表 7-2 環境変数一覧

環境変数	説明
PATH	実行形式のサーチ・パス
INC78K0R	インクルード・ファイルのサーチ・パス
TMP	テンポラリ・ファイルのサーチ・パス
LANG78K	漢字コードの種類（-zs，-ze，-zn オプションでも指定可能） （sjis：シフト JIS コード，euc：EUC コード，none：漢字コードなし）
LIB78K0R	ライブラリのサーチ・パス

【使用例】

＜コマンド・プロンプト使用時の場合＞

```
; AUTOEXEC.BAT
PATH C:¥Program Files¥NEC Electronics
Tools¥CC78K0R¥V x.xx ¥bin;C:¥bat;C:¥cc78k0r;C:¥tool
VERIFY ON
BREAK ON
SET INC78K0R=C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V x.xx ¥inc78k0r
SET LIB78K0R=C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V x.xx ¥lib78k0r
SET TMP=C:¥tmp
SET LANG78K=sjis
```

【説明】

- パス指定により，C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx*¥bin，C:¥bat，C:¥cc78k0r，C:¥tool という順で，実行形式ファイルを検索します。
- インクルード・ファイルは，C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx*¥inc78k0r から検索されます。
  設定しない場合，C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx*¥inc78k0r（C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx* にインストールした場合）から検索します。
- ライブラリ・ファイルは，リンク時に C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx*¥lib78k0r から検索します。
  設定しない場合，C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx*¥lib78k0r（C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx* にインストールした場合）から検索します。
- テンポラリ・ファイルは，C:\tmp に作成されます。
- 漢字コードはシフト JIS コードになります。

【注意】

PM+ の利用時に，環境変数の設定は行わないでください。

7.3 コンパイルを中断する

コマンド行でコンパイルを行う場合は，コマンド・キー入力（[CTRL] キー＋ [C] キー）により，コンパイルを中断することができます。break on を指定した場合には，キー入力のタイミングに関係なく，また，break off の場合には画面表示中のみに制御を OS に戻します。そして，オープン中のすべてのテンポラリ・ファイル，出力ファイルを削除します。

PM+ 上でビルド（メイク）を中止したい場合は，PM+ 上の [ビルド] メニュー→ [ビルドの中止] を選択するか，ツールバーの [ビルドの中止] ボタンを押してください。PM+ 上でのビルド時は，コマンド・キー入力は受け付けられません。

第 8 章 スタートアップ・ルーチン

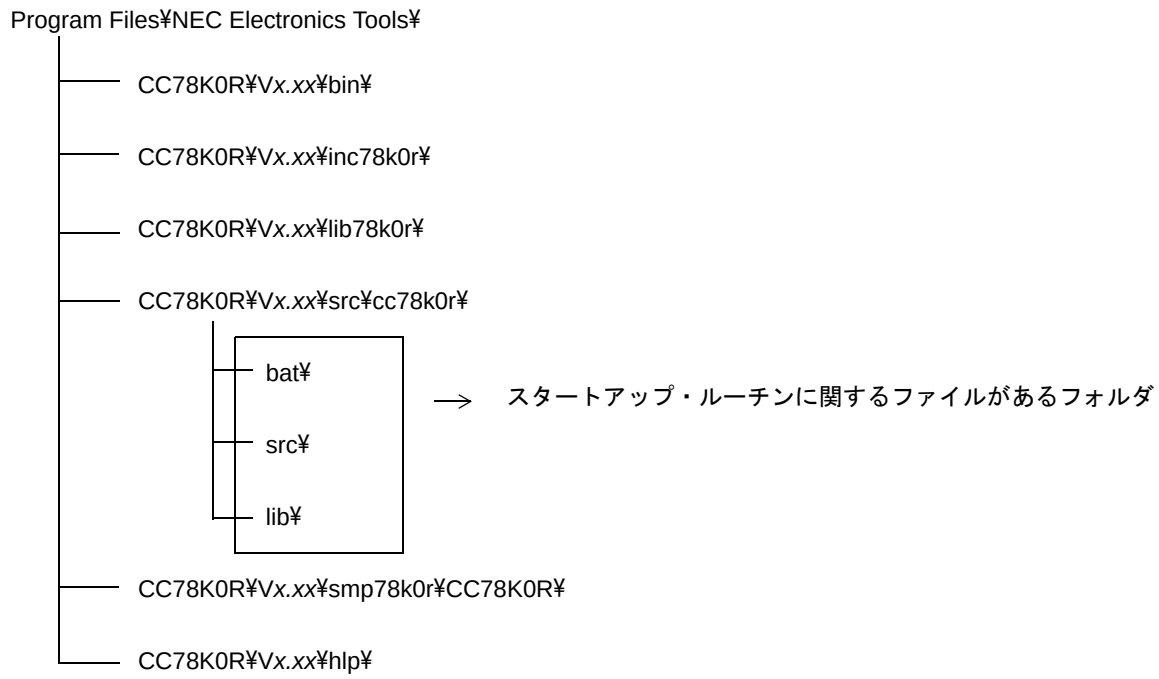
C 言語によるプログラムを実行させるには，システムへ組み込むための ROM 化処理，ユーザ・プログラム（main 関数）の起動などを行うプログラムが必要となります。このプログラムのことをスタートアップ・ルーチンと呼びます。

ユーザが作成したプログラムを実行させるためには，そのプログラムに応じたスタートアップ・ルーチンを作成しなければなりません。CC78K0R は，プログラム実行前に必要な処理を含むスタートアップ・ルーチンのオブジェクト・ファイルと，ユーザがシステムに合わせて変更できるようにスタートアップ・ルーチンのソース・ファイル（アセンブリ・ソース）を提供しています。スタートアップ・ルーチンのオブジェクト・ファイルをユーザ・プログラムとリンクすることにより，ユーザが実行前処理を記述しなくても実行可能なプログラムを作成することができます。

この章では，スタートアップ・ルーチンの内容，使い方，改良のポイントなどについて説明します。

8.1 ファイルの構成

スタートアップ・ルーチンに関するファイルは，C コンパイラ・パッケージのフォルダ src\cc78k0r に格納されています。



次に，src\cc78k0r 以下にあるフォルダの内容について示します。

8.1.1 フォルダ bat の内容

このフォルダのバッチ・ファイルは，PM+ 上では使用することができません。

これらのバッチ・ファイルは，スタートアップ・ルーチンなどのソース修正が必要な場合のみ使用してください。

表 8-1 フォルダ“bat”の内容

バッチ・ファイル名	説明
mkstup.bat	スタートアップ・ルーチンのアセンブル用バッチ・ファイル
reprom.bat	rom.asm 更新用バッチ・ファイル注 1
repgetc.bat	getchar.asm 更新用バッチ・ファイル
repputc.bat	putchar.asm 更新用バッチ・ファイル
repputcs.bat	_putchar.asm 更新用バッチ・ファイル
repselo.bat	setjmp.asm，longjmp.asm 更新用バッチ・ファイル （コンパイラ予約領域退避あり）注 2
repselon.bat	setjmp.asm，longjmp.asm 更新用バッチ・ファイル （コンパイラ予約領域退避なし）注 2
repvect.bat	vect*.asm 更新用バッチ・ファイル

注 1 ROM 化ルーチンは，ライブラリに含まれているため，このバッチ・ファイルでライブラリも更新されます。

注 2 コンパイラ予約領域（KREGxx などのために確保される saddr 領域）の退避がある setjmp/longjmp と，退避がない（レジスタの退避のみ行う）setjmp/longjmp を作成します。

8.1.2 フォルダ src の内容

フォルダ src には，スタートアップ・ルーチン，ROM 化ルーチン，エラー処理ルーチン，標準ライブラリ関数（一部）のアセンブラ・ソースが入っています。システムに合わせて修正が必要な場合は，このアセンブラ・ソースを修正し，bat フォルダのバッチ・ファイルでアセンブルなどを行うことにより，リンクするオブジェクト・ファイルを作成することができます。

表 8-2 フォルダ“src”の内容

スタートアップ・ルーチン・ソース・ファイル名	説明
cstart.asm 注	スタートアップ・ルーチンのソース・ファイル（標準ライブラリ使用時用）
cstartn.asm 注	スタートアップ・ルーチンのソース・ファイル（標準ライブラリ未使用時用）
rom.asm	ROM 化ルーチンのソース・ファイル
_putchar.asm	_putchar 関数
putchar.asm	putchar 関数
getchar.asm	getchar 関数
longjmp.asm	longjmp 関数
setjmp.asm	setjmp 関数
vectxx.asm	各割り込みベクタ・ソース（xx：ベクタ・アドレス）
def.inc	ライブラリ種別設定用
macro.inc	各種定型パターンについてのマクロ定義
vect.inc	フラッシュ領域分岐テーブルの先頭アドレス
library.inc	明示的にブート領域に配置するライブラリの選択

注 ファイル名に“n”が付加されたものは，標準ライブラリ処理がないスタートアップ・ルーチンです。標準ライブラリを使用しない場合のみに使用してください。
また，cstartb*.asm はブート領域用スタートアップ・ルーチン，cstarte*.asm はフラッシュ領域用スタートアップ・ルーチンです。

8.1.3 フォルダ lib の内容

フォルダ lib には，スタートアップ・ルーチン，ライブラリのソースをアセンブルしたオブジェクト・ファイルが入っています。このオブジェクト・ファイルは，78K0R シリーズであれば，どのターゲット・デバイス用のプログラムとでもリンクすることができます。特に修正が必要ない場合には，あらかじめ入っているオブジェクト・ファイルをそのままリンクしてください。CC78K0R が提供している mkstup.bat を実行すると，このオブジェクト・ファイルは上書きされます。

表 8-3 フォルダ“lib”の内容

ファイル名			説明
通常	ブート領域	フラッシュ領域	
cl0rm.lib cl0rl.lib cl0rmf.lib cl0rlf.lib cl0rxm.lib cl0rxl.lib	cl0rm.lib cl0rl.lib cl0rmf.lib cl0rlf.lib cl0rxm.lib cl0rxl.lib	cl0rme.lib cl0rle.lib cl0rmfe.lib cl0rlfe.lib cl0rxme.lib cl0rxle.lib	ライブラリ・ファイル（ランタイム・ライブラリ，標準ライブラリ）
s0rm.rel s0rml.rel s0rl.rel s0rll.rel	s0rmb.rel s0rmlb.rel s0rlb.rel s0rllb.rel	s0rme.rel s0rmle.rel s0rle.rel s0rlle.rel	スタートアップ・ルーチンのオブジェクト・ファイル

ファイル内容の詳細については，「2.5.1 ライブラリ・ファイル」を参照してください。

8.2 バッチ・ファイルの説明

8.2.1 スタートアップ・ルーチン作成用バッチ・ファイル

スタートアップ・ルーチンのオブジェクト・ファイルを作成するには，フォルダ bat にある mkstup.bat を使用します。

また，mkstup.bat では，RA78K0R　アセンブラ・パッケージの中のアセンブラが必要となります。したがって，PATH を設定していない場合は，設定して動作できるようにしてください。

次に，使用方法を示します。

【使用方法】

- mkstup.bat のあるフォルダ src\cc78k0r\bat で，次のようにコマンド行で実行してください。

```
mkstup デバイス種別注
```

注 各デバイスのユーザーズ・マニュアル，または「デバイス・ファイル　使用上の留意点」を参照してください。

【使用例】

- 対象品種が uPD78F1166_A0 のときに使用するスタートアップ・ルーチンを作成します。

```
mkstup f1166a0
```

バッチ・ファイル mkstup.bat は，次のようにフォルダ bat と同じ階層のフォルダ lib の下にスタートアップ・ルーチンのオブジェクト・ファイルを上書きする形で格納します。

それぞれのフォルダには，オブジェクト・ファイルをリンクするときに必要なスタートアップ・ルーチンが出力されます。

次に，lib に作成されるオブジェクト・ファイル名を示します。

lib ――― s0rm.rel
s0rmb.rel
s0rme.rel
s0rml.rel
s0rmlb.rel
s0rmle.rel
s0rl.rel
s0rlb.rel
s0rle.rel
s0rll.rel
s0rllb.rel
s0rlle.rel

8.3 スタートアップ・ルーチン

8.3.1 スタートアップ・ルーチンの概要

スタートアップ・ルーチンは，ユーザが作成した C ソース・プログラムを実行させるために必要な準備を行います。ユーザのプログラムとリンクさせることにより，目的を果たすロード・モジュール・ファイルを作成することができます。

(1) 機能

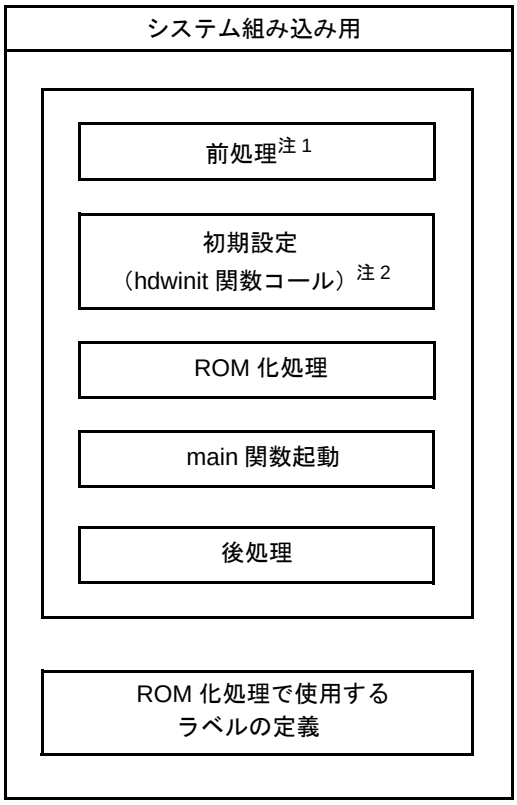
メモリの初期化，システムへ組み込むための ROM 化処理，C ソース・プログラムの起動，終了処理などを行います。

- ROM 化処理

C ソース・プログラム中で定義された外部変数，スタティック変数，sreg 変数の初期値は，ROM に配置されます。しかし，ROM に配置されたままでは，変数の値を書き換えることができません。そのため，ROM に配置された初期値を RAM にコピーする必要があります。この処理を ROM 化処理といい，プログラムを ROM に書き込んだとき，マイコン上で動作できるようにします。

(2) 構成

スタートアップ・ルーチンに関連するプログラムとその構成を，次に示します。



注 1 標準ライブラリを使用する場合は，ライブラリに関する処理が最初に行われます。スタートアップ・ルーチン・ソース・ファイルの名前の最後に n がないものは，標準ライブラリに関する処理があり，n が付いたファイルは処理が省かれています。

注 2 hdwinit 関数は，周辺装置（sfr）の初期設定をする関数としてユーザが必要に応じて作成する関数です。hdwinit 関数を作成することにより，初期設定のタイミングを早くすることができます（main 関数の中でも初期設定は可能です）。ユーザが hdwinit 関数を作成しない場合は，何もせずにリターンします。

cstart.asm，と cstartn.asm は，ほぼ同じ内容です。
上記のファイルの違いを，次に示します。

スタートアップ・ルーチンの種類	ライブラリ処理の有無
cstart.asm	あり
cstartn.asm	なし

(3) スタートアップ・ルーチンの使い分け

CC78K0R が提供している各ソース・ファイルに対応したオブジェクト・ファイル名を，次に示します。

ファイルの種類	ソース・ファイル	オブジェクト・ファイル
スタートアップ・ルーチン	cstart*.asm 注 1，2	s0r*.rel 注 2，3，4
ROM 化ファイル	rom.asm	ライブラリに含まれます。

注 1 *：標準ライブラリを使用しない場合は“n”が付きます。使用する場合は文字は付きません。

注 2 *：ブート領域用の場合は“b”，フラッシュ領域用の場合は“e”が付きます。

注 3 *：標準ライブラリの固定領域を使用する場合は“ l ”が付きます。

注 4 *：スモール・モデル，ミディアム・モデルの場合は“m”，コンパクト・モデル，ラージ・モデルの場合は“l”が付きます。
スモール・モデル，ミディアム・モデルの場合でも，far 領域に変数を配置する場合は，“l”付きのスタートアップ・ルーチンを使用してください。

備考 rom.asm は，ROM 化処理でコピーされるデータの最終アドレスを示すラベルを定義しています。
rom.asm のオブジェクトは，ライブラリに含まれています。

8.3.2 サンプル・プログラム（cstart.asm）の説明

ここでは，スタートアップ・ルーチンの内容について，cstart.asm，rom.asm を例に説明します。スタートアップ・ルーチンは，前処理，初期設定，ROM 化処理，main 関数の起動と後処理から構成されています。

備考 cstart などは，先頭に _@ を付加した形式で呼び出されます。

(1) 前処理

cstart.asm の前処理について説明します。

＜ cstart.asm の前処理＞

```
        NAME    @cstart

$INCLUDE ( def.inc )                                                    ; (1)
$INCLUDE ( macro.inc )
                                                                        ; (2)
BRKSW     EQU   1 ; brk , sbrk , calloc , free , malloc , realloc function use
EXITSW    EQU   1     ; exit , atexit function use
RANDSW    EQU   1     ; rand , srand  function use
DIVSW     EQU   1     ; div           function use
LDIVSW    EQU   1     ; 1div          function use
FLOATSW   EQU   1     ; floating point variables use
STRTOKSW  EQU   1     ; strtok        function use

        PUBLIC  _@cstart , _@cend                                       ; (3)

$_IF ( BRKSW )
        PUBLIC  _@BRKADR , _@MEMTOP , _@MEMBTM
          :
$ENDIF
        EXTRN   _main , _@STBEG , _hdwinit , _@MAA                      ; (4)
$_IF ( EXITSW )
        EXTRN   _exit
$ENDIF
                                                                        ; (5)
        EXTRN   _?R_INIT , _?RLINIT , _?R_INIS , _?DATA , _?DATAL , _?DATS
@@DATA  DSEG    BASEP ; near                                            ; (6)

$_IF ( EXITSW )
_@FNCTBL :      DS    4 * 32
_@FNCENT :      DS    2
  :
_@MEMTOP :      DS    32
_@MEMBTM :
$ENDIF
```

(1) インクルード・ファイルの取り込み

def.inc → ライブラリ種別設定用

macro.inc → 各種定型パターンについてのマクロ定義

(2) ライブラリ・スイッチ

コメントにある標準ライブラリを使用しない場合は，EQU 定義を 0 に修正することにより，使用しないライブラリの処理や，ライブラリ用に確保している領域を節約できます。デフォルトは，すべて使用する設定になっています（ライブラリ処理なしのスタートアップ・ルーチンには，この処理はありません）。

(3) シンボル定義

標準ライブラリ使用時に使用するシンボルを定義します。

(4) スタック解決用のシンボルの外部参照宣言

- スタック解決用のパブリック・シンボル（_@STBEG）を外部参照宣言します。
  _@STBEG は，スタック領域の最終アドレス +1 を値として持ちます。
- _@STBEG は，リンカのスタック解決用シンボル生成オプション -s を指定することにより，自動生成されます。したがって，リンク時には -s オプションを必ず指定してください。この際，スタックに使用する領域名も指定してください。領域名を省略した場合は，RAM という領域を使用しますが，リンク・ディレクティブ・ファイルを作成することにより，スタック用領域を自由に配置することができます。メモリ・マップに関しては，ターゲット・デバイスのユーザーズ・マニュアルを参照してください。

次にリンク・ディレクティブ・ファイルの例を示します。リンク・ディレクティブ・ファイルは，通常のエディタでユーザが作成するテキスト・ファイルです（記述方法に関する詳細については，「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアルを参照してください）。

＜例：リンク時に -sSTACK を指定した場合＞

lk78k0r.dr（リンク・ディレクティブ・ファイル）を作成します。ターゲット・デバイスのメモリ・マップを参照して ROM，RAM はデフォルトで配置されるので，配置を変更しない場合は，指定する必要はありません。
リンク・ディレクティブについては，smp78k0r\CC78K0R フォルダの lk78k0r.dr を参考にしてください。

```
                  先頭アドレス    サイズ
                       ↓          ↓
memory  SDR     : ( 0xFFE20h , 0000098h )
memory  STACK   : ( 0xxxxxxh , 0xxxxxxh ) ← ここに先頭アドレスとサイズを
                                             指定し，-d リンカ・オプションで
                                             lk78k0r.dr を指定します
                                             （例：-dlk78k0r.dr）
merge @@INIS    : = SDR
merge @@DATS    : = SDR
merge @@BITS    : = SDR
```

(5) ROM 化処理用ラベルの外部参照宣言

ROM 化処理用ラベルは，後処理の部分で定義されます。

(6) 標準ライブラリ用領域確保

標準ライブラリ使用時に使用するための領域を確保します。

(2) 初期設定

cstart.asm の初期設定について説明します。

＜ cstart.asm の初期設定＞

```
@@VECT00        CSEG    AT      0                                       ; (1)
        DW      _@cstart

@@LCODE CSEG    BASE
_@cstart :
        SEL     RB0                                                     ; (3)
        MOV     A , #_@MAA                                              ; (2)
        MOV1    CY , A.0
        MOV1    MAA , CY
        MOVW    SP , #LOWW _@STBEG      ; SP <-stack begin address      ; (4)
        CALL    !!_hdwinit                                              ; (5)
          :
$_IF ( BRKSW OR EXITSW OR RANDSW OR FLOATSW )
        CLRW    AX
$ENDIF
          :
```

(1) リセット・ベクタの設定

リセット・ベクタ・テーブルのセグメントを次のように定義し，スタートアップ・ルーチンの先頭アドレスを設定します。

```
@@VECT00        CSEG    AT      0000H
                DW      _@cstart
```

(2) ミラー領域の設定

ミラー領域の設定を行います。

ミラー領域については，ターゲット・デバイスのユーザーズ・マニュアルを参照してください。

(3) レジスタ・バンクの設定

レジスタ・バンク RB0 をワーク・レジスタとして設定します。

(4) SP（スタック・ポインタ）の設定

スタック・ポインタに，_@STBEG を設定します。

_@STBEG は，リンカのスタック解決用シンボル生成オプション -s を指定することにより，自動生成されます。

(5) ハードウエア・イニシャライズ関数呼び出し

hdwinit 関数は，周辺装置（SFR）の初期設定をする関数としてユーザが必要に応じて作成する関数です。この関数を作成することにより，ユーザの目的に合った初期設定が可能となります。

ユーザが hdwinit 関数を作成しない場合は，何もせずリターンします。

(3) ROM 化処理

cstart.asm の ROM 化処理について説明します。

＜ ROM 化処理＞

```
; copy external variables having initial value
$_IF ( _ESCOPY )
        MOV     ES , #HIGHW _@R_INIT
$ENDIF
        MOVW    HL , #LOWW _@R_INIT
        MOVW    DE , #LOWW _@INIT
        BR      $LINIT2
LINIT1 :
$_IF ( _ESCOPY )
        MOV     A , ES : [ HL ]
$ELSE
        MOV     A , [ HL ]
$ENDIF
        MOV     [ DE ] , A
        INCW    HL
        INCW    DE
LINIT2 :
        MOVW    AX , HL
        CMPW    AX , #LOWW _?R_INIT
        BNZ     $LINIT1
```

ROM 化処理では，ROM に配置された外部変数，sreg 変数の初期値を RAM にコピーします。処理される変数は，次の例に示すように（a）～（d）の 4 種類あります。

＜例＞

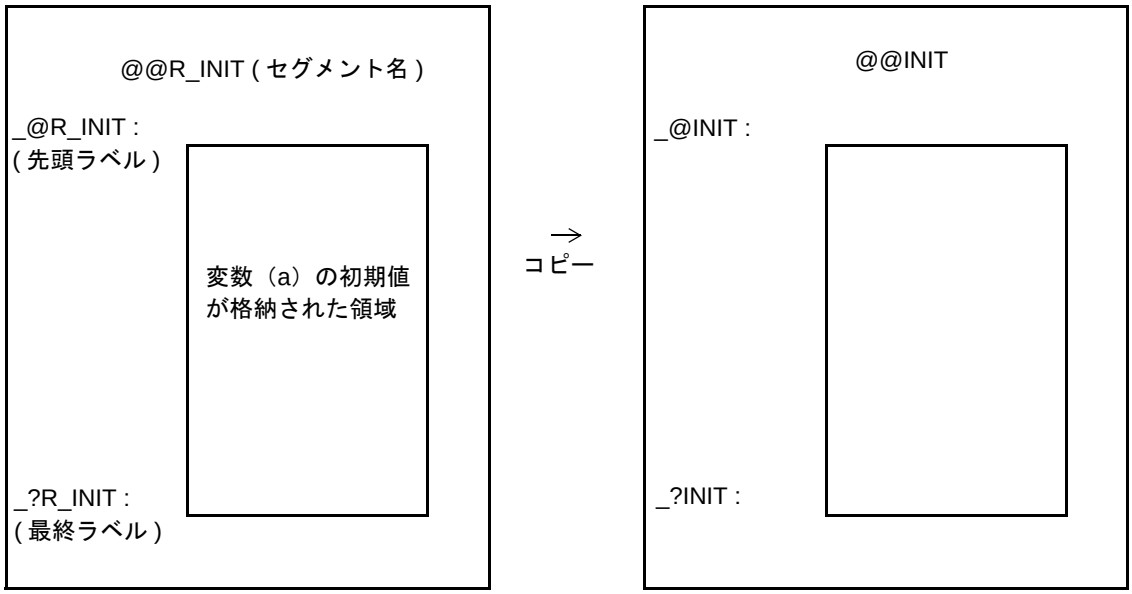
```
char    c = 1 ;                 （a）初期値あり外部変数
int     i ;                     （b）初期値なし外部変数注
__sreg  int     si = 0 ;        （c）初期値あり sreg 変数
__sreg  char    sc ;            （d）初期値なし sreg 変数注

main ( )
{
          :
}
```

注　初期値なし外部変数，sreg 変数はコピーせず，直接 RAM に 0 を入れます。

- （a）初期値あり外部変数の ROM 化処理を，次に示します。

  変数（a）の初期値は，CC78K0R により，ROM 上の @@R_INIT というセグメントに配置されます。

  ROM 化処理は，これらの値を RAM 上の @@INIT というセグメントにコピーします（変数（c）についても，同様な処理を行います）。



- @@R_INIT セグメントの先頭ラベル，最終ラベルは，_@R_INIT，_?R_INIT で，@@INIT セグメントの先頭ラベル，最終ラベルは，_@INIT，_?INIT で定義されています。

- 変数（b），（d）については，コピーではなく直接 RAM の決められたセグメントへ 0 を入れます。なお，（a）～（d）の変数が配置される ROM，RAM 領域のセグメント名，および各セグメントでの初期値の先頭，最終ラベルを，次に示します。

＜初期値の ROM 領域＞

変数の種類	セグメント	先頭ラベル	最終ラベル
初期値あり外部変数（a） （near 領域に配置の場合）	@@R_INIT	_@R_INIT	_?R_INIT
初期値あり外部変数（a） （far 領域に配置の場合）	@@RLINIT	_@RLINIT	_?RLINIT
初期値あり sreg 変数（c）	@@R_INIS	_@R_INIS	_?R_INIS

＜初期値の RAM 領域（コピー先）＞

変数の種類	セグメント	先頭ラベル	最終ラベル
初期値あり外部変数（a） （near 領域に配置の場合）	@@INIT	_@INIT	_?INIT
初期値あり外部変数（a） （far 領域に配置の場合）	@@INITL	_@INITL	_?INITL
初期値なし外部変数（b） （near 領域に配置の場合）	@@DATA	_@DATA	_?DATA
初期値なし外部変数（b） （far 領域に配置の場合）	@@DATAL	_@DATAL	_?DATAL
初期値あり sreg 変数（c）	@@INIS	_@INIS	_?INIS
初期値なし sreg 変数（d）	@@DATS	_@DATS	_?DATS

(4) main 関数の起動と後処理

cstart.asm の main 関数の起動と後処理について説明します。

＜ main 関数の起動と後処理＞

```
            CALL    !!_main          ; main ( ) ;            ; (1)
$_IF ( EXITSW )
            CLRW    AX
            CALL    !!_exit          ; exit ( 0 ) ;          ; (2)
$ENDIF
            BR      $$
;
_@cend :
                                                            ; (3)
@@R_INIT    CSEG    UNIT64KP
_@R_INIT :
@@RLINIT    CSEG    UNIT64KP
_@RLINIT :
@@R_INIS    CSEG    UNIT64KP
_@R_INIS :
@@INIT      DSEG    BASEP
_@INIT :
@@INITL     DSEG    UNIT64KP
_@INITL :
@@DATA      DSEG    BASEP
_@DATA :
@@DATAL     DSEG    UNIT64KP
_@DATAL :
@@INIS      DSEG    SADDRP
_@INIS :
@@DATS      DSEG    SADDRP
_@DATS :
@@CALT      CSEG    CALLT0
@@CNST      CSEG    MIRRORP
@@CNSTL     CSEG    PAGE64KP
@@BITS      BSEG
;
            END
```

(1) main 関数の起動

main 関数を呼び出します。

(2) exit 関数の起動

exit 処理が必要な場合は，exit 関数を呼び出します。

(3) ROM 化処理で使用するセグメント，ラベルの定義

ROM 化処理で，（a）～（d）の変数（「(3) ROM 化処理」を参照）ごとに，使用するセグメント，ラベルを定義します。セグメントは，各変数の初期値を格納する領域を示します。ラベルは，各セグメントの先頭アドレスを示します。

ROM 化用ファイル rom.asm について説明します。rom.asm のリロケータブル・オブジェクト・ファイルはライブラリの中に入っています。

```
        NAME    @rom
;
        PUBLIC  _?R_INIT , _?RLINIT , _?R_INIS
        PUBLIC  _?INIT , _?INITL , _?DATA , _?DATAL , _?INIS , _?DATS
;
@@R_INIT        CSEG    UNIT64KP                        ; (1)
_?R_INIT :
@@RLINIT        CSEG    UNIT64KP
_?RLINIT :
@@R_INIS        CSEG    UNIT64KP
_?R_INIS :
@@INIT          DSEG    BASEP
_?INIT :
@@INITL         DSEG    UNIT64KP
_?INITL :
@@DATA          DSEG    BASEP
_?DATA :
@@DATAL         DSEG    UNIT64KP
_?DATAL :
@@INIS          DSEG    SADDRP
_?INIS :
@@DATS          DSEG    SADDRP
_?DATS :
;
        END
```

(1) ROM 化処理で使用するラベルの定義

ROM 化処理で，（a）～（d）の変数（「(3) ROM 化処理」を参照）ごとに，使用するラベルを定義します。これらのラベルは，各変数の初期値を格納するセグメントの最終アドレスを示します。

8.3.3 スタートアップ・ルーチンなどの修正

CC78K0R が提供しているスタートアップ・ルーチンは，実際に使用するターゲット・システムに合わせて修正できます。ここでは，これらのファイルを修正する場合のポイントについて説明します。

(1) スタートアップ・ルーチンを修正する場合

スタートアップ・ルーチン・ソース・ファイルの修正のポイントについて説明します。修正後は，修正したソース・ファイル（cstart*.asm）を，フォルダ src\cc78k0r\bat にあるバッチ・ファイル mkstup.bat を用いて，アセンブルしてください（*：英数字）。

- ライブラリ関数で使われるシンボル

次の表で示されるライブラリ関数を使わないのであれば，スタートアップ・ルーチン（cstart.asm）中の各関数に対応するシンボルは削除することができます。ただし，exit 関数は，スタートアップ・ルーチンで使用されるので，_@FNCTBL，_@FNCENT を削除することはできません（exit 関数も削除する場合は，それらのシンボルも削除することができます）。使用しないライブラリ関数のシンボルなどについては，ライブラリ・スイッチを変更することで削除することができます。

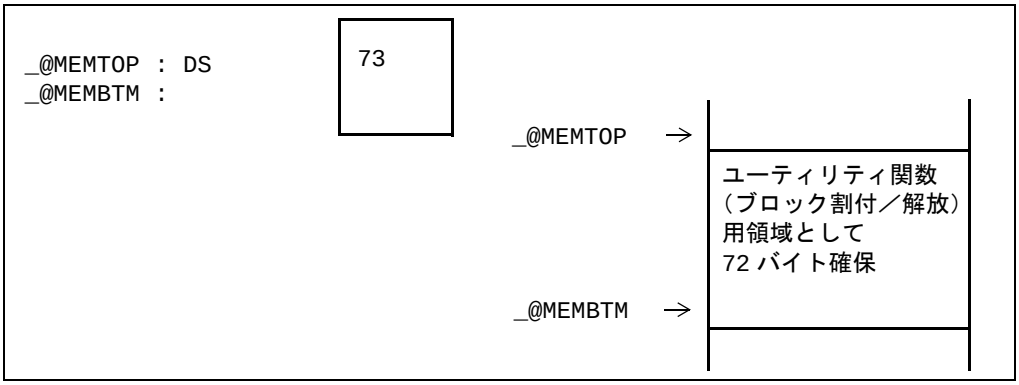
ライブラリ関数名	使われるシンボル
brk sbrk malloc calloc realloc free	_errno _@MEMTOP _@MEMBTM _@BRKADR
exit	_@FNCTBL _@FNCENT
rand srand	_@SEED
div	_@DIVR
ldiv	_@LDIVR
strtok	_@TOKPTR
atof strtod 数学関数 浮動小数点ランタイムライブラリ	_errno

- ユーティリティ関数（ブロック割付／解放）で使われる領域

ユーティリティ関数（ブロック割付／解放）で使われる領域サイズをユーザが定義する場合は，次の例のように設定します。

【例】

ユーティリティ関数（ブロック割付／解放）用として，72 バイト確保したい場合，スタートアップ・ルーチンの初期設定を，次のように修正してください。

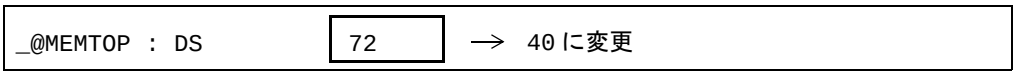


スタートアップ・ルーチンに指定する数値は，確保したい領域サイズに 1 バイトを加えた値としてください。この例では，スタートアップ・ルーチンで 73 バイト確保していますが，実際にユーティリティ関数では 72 バイトまで確保可能となります。

指定したサイズが大きすぎる場合，RAM 領域に入りきらず，リンク時にエラーとなることがあります。このような場合には，次のように指定するサイズを小さくするか，リンク・ディレクティブ・ファイルを修正して回避してください。リンク・ディレクティブ・ファイルを修正する場合は，「(2) リンク・ディレクティブ・ファイル」を参照してください。

【例】

指定するサイズを小さくする場合



(2) リンク・ディレクティブ・ファイル

リンク・ディレクティブ・ファイルの作成方法を説明します。実際のターゲット・システムに合わせて作成し，リンク時に -d オプションで作成したファイルを指定してください。

なお，作成の際には，次のことに注意してください（リンク・ディレクティブの詳しい記述方法については，「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアルを参照してください）。

- CC78K0R は，ショート・ダイレクト・アドレス領域（saddr 領域）の一部を，次のような CC78K0R 固有の目的で使用する場合があります。

  具体的には，FFEB4H - FFEDFH までの 44 バイトの領域です。

  (a) -qr オプションを指定した場合の register 変数 [FFEB4H - FFEC3H]

  (b) norec 関数の引数，自動変数 [FFEC4H - FFED3H]

  (c) セグメント情報 [FFED4H - FFED7H]

  (d) ランタイム・ライブラリの引数 [FFED8H - FFEDFH]

  (e) 標準ライブラリの作業用 （(a) と (b) の領域の一部）

  注意 ユーザが標準ライブラリを使用しない場合，(e) の領域は使用されません。

次にリンク・ディレクティブ・ファイル（lk78k0r.dr）で RAM サイズを変更する例を示します。

メモリ・サイズの変更をする場合は，他の領域と重ならないように注意してください。変更の際には，使用するターゲット・デバイスのメモリ・マップを参照してください。

＜ lk78k0r.dr ＞

```
                    先頭アドレス  サイズ
memory RAM :    ( 0FCF00h , 002F20h ) → このサイズを大きくします。
memory SDR :    ( 0FFE20h , 000098h )    （必要に応じて，先頭アドレスも変更します。）
merge @@INIS : = SDR                   → セグメントの配置を指定しています。
merge @@DATS : = SDR                   → セグメントの配置を指定しています。
merge @@BITS : = SDR                   → セグメントの配置を指定しています。
```

セグメントの配置を変更したい場合は，merge 文を追加します。コンパイラ出力セクション名の変更機能を使用した場合，セグメントを独自に配置することができます（詳細については，「CC78K0R C コンパイラ 言語編」のユーザーズ・マニュアルを参照してください）。

セグメントの配置を変更した結果，配置するメモリが足りなくなった場合は，対応する memory 文を変更してください。

(3) RTOS を使用する場合

RX78K0R，および CC78K0R は，初期化処理ルーチン（アセンブラ形式）のサンプルをそれぞれ提供しています。したがって，RX78K0R と CC78K0R を併用する場合には，双方の初期化処理ルーチンの修正が必要となります。

初期化処理ルーチンの修正方法については，「RX78K0R 機能編」のユーザーズ・マニュアルを参照してください。

8.4 フラッシュ領域用スタートアップ・モジュールでの ROM 化処理

フラッシュ用スタートアップ・モジュールでは，通常のスタートアップ・モジュールと次の点が異なります。

表 8-4 初期化データの ROM 領域のセクション

変数の種類	セグメント	先頭ラベル	終端ラベル
初期値あり外部変数（a） （near 領域に配置の場合）	@ER_INIT CSEG UNIT64KP	E@R_INIT	E?R_INIT
初期値あり外部変数（a） （far 領域に配置の場合）	@ERLINIT CSEG UNIT64KP	E@RLINIT	E?RLINIT
初期値あり sreg 変数（c）	@ER_INIS CSEG UNIT64KP	E@R_INIS	E?R_INIS

表 8-5 コピー先の RAM 領域のセクション

変数の種類	セグメント	先頭ラベル	終端ラベル
初期値あり外部変数（a） （near 領域に配置の場合）	@EINIT DSEG BASEP	E@INIT	E?INIT
初期値あり外部変数（a） （far 領域に配置の場合）	@EINITL DSEG UNIT64KP	E@INITL	E?INITL
初期値なし外部変数（b） （near 領域に配置の場合）	@EDATA DSEG BASEP	E@DATA	E?DATA
初期値なし外部変数（b） （far 領域に配置の場合）	@EDATAL DSEG UNIT64KP	E@DATAL	E?DATAL
初期値あり sreg 変数（c）	@EINIS DSEG SADDRP	E@INIS	E?INIS
初期値なし sreg 変数（d）	@EDATS DSEG SADDRP	E@DATS	E?DATS

- スタートアップ・モジュールでは，ROM 領域，RAM 領域の各セグメントの先頭としてそれぞれに次のラベルを付けます。

  E@R_INIT，E@R_INIS，E@INIT，E@DATA，E@INIS，E@DATS，E@INITL，E@DATAL

  コンパクト・モデル，ラージ・モデルの場合，または far 領域に変数を配置する場合，さらに次のラベルを付けます。

  E@RLINIT，E@INITL，E@DATAL

- 終端モジュールでは，ROM 領域，RAM 領域の各セグメントの終端としてそれぞれに次のラベルを付けます。

  E?R_INIT，E?R_INIS，E?INIT，E?DATA，E?INIS，E?DATS，E?RLINIT，E?INITL，E?DATAL

- スタートアップ・モジュールは ROM 領域の各セグメントの先頭ラベルのアドレスから，終端ラベルのアドレス -1 までの内容を，RAM 領域の各セグメントの先頭ラベルのアドレスからの領域にコピーします。

- E@DATA から E?DATA まで，E@DATS から E?DATS まで，0 を埋め込みます。

- コンパクト・モデル，ラージ・モデルの場合，または far 領域に変数を配置する場合，さらに E@DATAL から E?DATAL まで，0 を埋め込みます。

第 9 章 エラー・メッセージ

この章では，CC78K0R が出力するエラー・メッセージについて説明します。

9.1 エラー・メッセージの形式

エラー・メッセージの形式は，次のとおりです。

```
ソース・ファイル名 ( 行番号 ) : エラー・メッセージ
```

＜例＞

```
prime.c ( 8 )   : CC78K0R error E0712 : Declaration syntax
prime.c ( 8 )   : CC78K0R error E0301 : Syntax error
prime.c ( 8 )   : CC78K0R error E0701 : External definition syntax
prime.c ( 19 )  : CC78K0R warning W0745 : Expected function prototype
```

ただし，C0101，C0103，C0104 の内部エラーのみ次の出力形式となります。

```
[ xxx.c < yyy > zzz ] CC78K0R error C0101 : Internal error
[ xxx.c < yyy > zzz ] CC78K0R error C0103 : Intermediate file error
[ xxx.c < yyy > zzz ] CC78K0R error C0104 : Illegal use of register
```

備考 *xxx*.c ： ソース・ファイル名

yyy ： 行番号

zzz ： メッセージ

9.2 エラー・メッセージの種類

CC78K0R が出力するエラー・メッセージには，次の 10 種類があります。

- コマンド行に対するエラー・メッセージ
- 内部エラー，メモリに対するエラー・メッセージ
- 文字に対するエラー・メッセージ
- 構成要素に対するエラー・メッセージ
- 変換に対するエラー・メッセージ
- 式に対するエラー・メッセージ
- 文に対するエラー・メッセージ
- 宣言，関数定義に対するエラー・メッセージ
- 前処理指令に対するエラー・メッセージ
- 致命的なファイル I/O，許されない OS 上での起動に対するエラー・メッセージ

9.3 エラー・メッセージ一覧

エラー・メッセージ一覧を活用する前に，エラー番号の形式を理解しておく必要があります。

エラー番号は，エラー・メッセージの種類とエラーに対する CC78K0R の処理を示しています。

エラー番号の形式は，次のようになります。

F/E/C/W*xxxx*

- アボート・エラー（F*xxxx*）

  必ずコンパイルを中止します。

  オブジェクト・モジュール・ファイル，アセンブラ・ソース・ファイルを出力しません。

- フェイタル・エラー（E*xxxx*）

  一定数以上発生した場合，コンパイルを中止します。

  オブジェクト・モジュール・ファイル，アセンブラ・ソース・ファイルを出力しません。

- 内部エラー（C*xxxx*）

  必ずコンパイルを中止します。

  オブジェクト・モジュール・ファイル，アセンブラ・ソース・ファイルを出力しません。

- ワーニング・エラー（W*xxxx*）

  コンパイルを続行します。

備考 *xxxx*（4 桁の数字）

種類	説明
0001 ～	コマンド行に対するエラー・メッセージ
0101 ～	内部エラー，メモリに対するエラー・メッセージ
0201 ～	文字に対するエラー・メッセージ
0301 ～	構成要素に対するエラー・メッセージ
0401 ～	変換に対するエラー・メッセージ
0501 ～	式に対するエラー・メッセージ
0601 ～	文に対するエラー・メッセージ
0701 ～	宣言，関数定義に対するエラー・メッセージ
0801 ～	前処理指令に対するエラー・メッセージ
0901 ～	致命的なファイル I/O，許されない OS 上での起動に対するエラー・メッセージ

注意 ファイル名に文法的誤りがあった場合には，メッセージにファイル名が付加されます。

エラー・メッセージは，開発する C コンパイラの言語仕様により追加，変更，削除することがあります。

9.3.1 コマンド行に対するエラー・メッセージ

表 9-1 コマンド行に対するエラー・メッセージ〈0001 ～〉

エラー番号	エラー・メッセージ	
F0001	メッセージ	Missing input file
	原因	入力ソース・ファイル名が指定されていません。
	ユーザの処置	Please enter 'cc78k0r --' if you want help message が出力されます。--/-?/-h オプションを使用し，オンライン・ヘルプ・ファイルなどを参照し，正しい入力を行ってください。
F0002	メッセージ	Too many input files
	原因	入力ソース・ファイル名が複数指定されています。
	ユーザの処置	Please enter 'cc78k0r --' if you want help message が出力されます。--/-?/-h オプションを使用し，オンライン・ヘルプ・ファイルなどを参照し，正しい入力を行ってください。
F0003	メッセージ	Unrecognized string
	原因	対話形式のコマンド行にオプション以外のものが指定されました。
F0004	メッセージ	Illegal file name ファイル名
	原因	指定されたファイル名として形式，文字，文字数のいずれかに誤りがあります。
F0005	メッセージ	Illegal file specification
	原因	ファイル名に不当なものが指定されました。
F0006	メッセージ	File not found
	原因	指定された入力ファイルが存在しません。
F0007	メッセージ	Input file specification overlapped ファイル名
	原因	入力ファイル名が重複して指定されました。
F0008	メッセージ	File specification conflicted ファイル名
	原因	入出力ファイル名が重複して指定されました。
F0009	メッセージ	Unable to make file ファイル名
	原因	指定された出力ファイルがリード・オンリー・ファイルとしてすでに存在しているため，作成することができません。
F0010	メッセージ	Directory not found
	原因	出力ファイル名中に存在しないドライブ，またはフォルダが含まれています。
F0011	メッセージ	Illegal path
	原因	パラメータにパス名を指定するオプションで，パス名以外が指定されました。

エラー番号	エラー・メッセージ	
F0012	メッセージ	Missing parameter ' オプション '
	原因	必要なパラメータが指定されていません。
	ユーザの処置	Please enter 'cc78k0r --' if you want help message が出力されます。--/-?/-h オプションを使用し，ヘルプ・ファイルなどを参照し，正しい入力を行ってください。
F0013	メッセージ	Parameter not needed ' オプション '
	原因	不要なオプション・パラメータが指定されました。
	ユーザの処置	Please enter 'cc78k0r --' if you want help message が出力されます。--/-?/-h オプションを使用し，ヘルプ・ファイルなどを参照し，正しい入力を行ってください。
F0014	メッセージ	Out of range ' オプション '
	原因	オプション・パラメータの指定数値が範囲外です。
	ユーザの処置	Please enter 'cc78k0r --' if you want help message が出力されます。--/-?/-h オプションを使用し，ヘルプ・ファイルなどを参照し，正しい入力を行ってください。
F0015	メッセージ	Parameter is too long
	原因	オプション・パラメータの文字数が制限を越えて指定されました。
F0016	メッセージ	Illegal parameter ' オプション '
	原因	オプション・パラメータの文法に誤りがあります。
	ユーザの処置	Please enter 'cc78k0r --' if you want help message が出力されます。--/-?/-h オプションを使用し，ヘルプ・ファイルなどを参照し，正しい入力を行ってください。
F0017	メッセージ	Too many parameters
	原因	オプション・パラメータの総数が制限を越えました。
F0018	メッセージ	Option is not recognized ' オプション '
	原因	誤ったオプションが指定されました。
	ユーザの処置	Please enter 'cc78k0r --' if you want help message が出力されます。--/-?/-h オプションを使用し，ヘルプ・ファイルなどを参照し，正しい入力を行ってください。
F0019	メッセージ	Parameter file nested
	原因	パラメータ・ファイル中に -f オプションが指定されました。
	ユーザの処置	パラメータ・ファイルの中に，パラメータ・ファイルを指定することはできないため，ネストしないように修正してください。
F0020	メッセージ	Parameter file read error
	原因	パラメータ・ファイルの読み込みに失敗しました。
F0021	メッセージ	Memory allocation failed
	原因	メモリ・アロケーションに失敗しました。

エラー番号	エラー・メッセージ	
W0022	メッセージ	Same category option specified - ignored ' オプション '
	原因	相反するオプションが重複して指定されました。
	プログラムの処理	あとに指定されたオプションを有効にして，処理を続けます。
W0023	メッセージ	Incompatible chip name
	原因	コマンド行上のデバイス種別とソース中のデバイス種別が異なります。
	プログラムの処理	コマンド行上のデバイス種別を優先します。
F0024	メッセージ	Illegal chip specifier on command line
	原因	コマンド行上のデバイス種別に誤りがあります。
W0029	メッセージ	'-QC' option is not portable
	原因	-qc オプションは，ANSI 準拠ではありません（-qc についての詳細は，「第 5 章 コンパイラ・オプション」を参照してください）。
W0031	メッセージ	'-ZP' option is not portable
	原因	-zp オプションは，ANSI 準拠ではありません（-zp についての詳細は，「第 5 章 コンパイラ・オプション」を参照してください）。
W0032	メッセージ	'-ZC' option is not portable
	原因	-zc オプションは，ANSI 準拠ではありません（-zc についての詳細は，「第 5 章 コンパイラ・オプション」を参照してください）。
F0033	メッセージ	Same category option specified ' オプション '
	原因	相反するオプションが重複して指定されました。
	ユーザの処置	Please enter 'cc78k0r --' if you want help message が出力されます。--/-?/-h オプションを使用し，ヘルプ・ファイルなどを参照し，正しい入力を行ってください。
W0046	メッセージ	'-ZF' option specified -regarded as '-QL1'
	原因	フラッシュ領域のオブジェクト作成オプション（-zf）が指定されたため，定型コード・パターンのライブラリ置き換えオプション（-ql）で，-ql2 以降は -ql1 とみなします。
W0067	メッセージ	' オプション ' option deleted - ignored
	原因	削除されたオプションが指定されました。‘オプション’ は無視されます。
W0068	メッセージ	' オプション 1' option deleted - regarded as ' オプション 2'
	原因	‘オプション 1’ は削除されたため，‘オプション 2’ を有効にします。

9.3.2 内部エラー，メモリに対するエラー・メッセージ

表 9-2 内部エラー，メモリに対するエラー・メッセージ〈0101 〜〉

エラー番号	エラー・メッセージ	
C0101	メッセージ	Internal error
	原因	内部エラーが起こりました。
	ユーザの処置	問い合わせ先にご連絡ください。
E0102	メッセージ	Too many errors
	原因	文法の誤りやコンパイルの制限によるエラーの合計が 30 を越えました。
	プログラムの処理	処理を継続しますが，これ以降のエラー・メッセージは出力しません。これ以前のエラーが多数のエラーを引き起こしている可能性があります。これ以前のエラーを最初に取り除いてください。
C0103	メッセージ	Intermediate file error
	原因	中間ファイルの内容に誤りがあります。
	ユーザの処置	問い合わせ先にご連絡ください。
C0104	メッセージ	Illegal use of register
	原因	レジスタの使い方に誤りがあります。
E0105	メッセージ	Register overflow : simplify expression
	原因	式が複雑すぎるので使用できるレジスタがなくなりました。
	ユーザの処置	エラーとなっている複雑な式を単純化してください。
C0106	メッセージ	Stack overflow ' オーバフロー要因 '
	原因	スタックのオーバフローが起こりました。 オーバフロー要因は stack，あるいは heap です。
	ユーザの処置	問い合わせ先にご連絡ください。
E0108	メッセージ	Compiler limit : too much automatic data in function
	原因	関数のオートマティック変数に割り当てられた領域が 64 K バイトの制限を越えました。
	ユーザの処置	64 K バイトを越えないように，変数を減らしてください。
E0109	メッセージ	Compiler limit : too much parameter of function
	原因	関数のパラメータに割り当てられた領域が 64 K バイトの制限を越えました。
	ユーザの処置	64 K バイトを越えないように，パラメータを減らしてください。
E0110	メッセージ	Compiler limit : too much code defined in file
	原因	ファイル内のコードに割り当てられた領域が，64 K バイトの制限を越えました。
E0111	メッセージ	Compiler limit : too much global data defined in file
	原因	ファイル内のグローバル変数に割り当てられた領域が 64 K バイトの制限を越えました。

エラー番号	エラー・メッセージ	
E0113	メッセージ	Compiler limit : too many local lables
	原因	1 関数内の内部ラベル数が処理限界数を越えました。
	ユーザの処置	関数本体が大き過ぎます。 関数を分割してください。

9.3.3 文字に対するエラー・メッセージ

表 9-3 文字に対するエラー・メッセージ〈0201 〜〉

エラー番号	エラー・メッセージ	
E0201	メッセージ	Unknown character '16 進数 '
	原因	指定された内部コードを持つ文字は認識することができません。
E0202	メッセージ	Unexpected EOF
	原因	関数の途中でファイルが終了しました。
W0203	メッセージ	Trigraph encountered
	原因	トライグラフ・シーケンス（3 文字表記）がありました。
	ユーザの処置	-za オプションを指定した場合は，トライグラフ・シーケンスが有効となるため，このワーニングは出力されません。

9.3.4 構成要素に対するエラー・メッセージ

表 9-4 構成要素に対するエラー・メッセージ〈0301 ～〉

エラー番号	エラー・メッセージ	
E0301	メッセージ	Syntax error
	原因	構文エラーが起こりました。
	ユーザの処置	ソースに記述ミスがないか確かめてください。
E0303	メッセージ	Expected identifier
	原因	goto 文の識別子が必要です。
	ユーザの処置	goto 文に指定する識別子を正しく記述してください。
W0304	メッセージ	Identifier truncate to ' 識別子 '
	原因	指定された識別子が長すぎます。識別子が“_”（アンダスコア）を含め 250 文字を越えました。
	ユーザの処置	識別子の長さを短くしてください。
E0305	メッセージ	Compiler limit : too many identifiers with block scope
	原因	1 つのブロック内でブロック・スコープを持つシンボルの数が多すぎます。
E0306	メッセージ	Illegal index , indirection not allowed
	原因	ポインタの値をとらない式に添字が使われています。
E0307	メッセージ	Call of non-function ' 変数名 '
	原因	変数名が関数名として使われています。
E0308	メッセージ	Improper use of a typedef name
	原因	typedef 名が正しく使われていません。
W0309	メッセージ	Unused ' 変数名 '
	原因	指定された変数はソース中で宣言されていますが，まったく使われていません。
W0310	メッセージ	' 変数名 ' is assigned a value which is never used
	原因	指定された変数は代入文には使われていますが，その他ではまったく使われていません。
E0311	メッセージ	Number syntax
	原因	定数の表現が誤っています。
E0312	メッセージ	Illegal octal digit
	原因	8 進数字としてふさわしくないものがあります。
E0313	メッセージ	Illegal hexadecimal digit
	原因	16 進数字としてふさわしくないものがあります。
E0314	メッセージ	Too big constant
	原因	定数が大きすぎて表現できません。

エラー番号	エラー・メッセージ	
E0315	メッセージ	Too small constant
	原因	定数が小さすぎて表現できません。
E0316	メッセージ	Too many character constants
	原因	文字定数が 2 文字を越えています。
E0317	メッセージ	Empty character constant
	原因	文字定数 ‘ ’ の中が空になっています。
E0318	メッセージ	No terminated string literal
	原因	文字列の終わりに ‘ " ’ がありません。
E0319	メッセージ	Changing string literal
	原因	文字列リテラルの書き換えを行っています。
W0320	メッセージ	No null terminator in string literal
	原因	文字列リテラル中にヌル文字を付加していません。
E0321	メッセージ	Compiler limit : too many characters in string literal
	原因	文字列リテラルの文字数が 509 を越えました。
E0322	メッセージ	Ellipsis requires three periods
	原因	C コンパイラは，“ .. ” を検出しましたが “...” である必要があります。
E0323	メッセージ	Missing ' 区切り子 '
	原因	区切り子に誤りがあります。
E0324	メッセージ	Too many }'s
	原因	“ { ” と “ } ” が正しく対応していません。
E0325	メッセージ	No terminated comment
	原因	コメントの終わりに “ */ ” がありません。
E0326	メッセージ	Illegal binary digit
	原因	2 進数としてふさわしくないものがあります。
E0327	メッセージ	Hex constants must have at least one hex digit
	原因	16 進型定数表記では，少なくとも 1 桁の 16 進数が必要です。
W0328	メッセージ	Unrecognized character escape sequence ' 文字 '
	原因	エスケープ・シーケンスとして正しく認識することができません。
E0329	メッセージ	Compiler limit : too many comment nesting
	原因	コメントのネストの数が 255 の制限を越えました。
W0332	メッセージ	Non-supported keyword found-ignored ' 関数属性子 ' in this file
	原因	サポートしないキーワードを検出しました。 ファイル中の関数属性子は無視します。

エラー番号	エラー・メッセージ	
W0340	メッセージ	Unreferenced label ' ラベル名 '
	原因	指定されたラベルは定義済みですが，一度も参照されていません。
E0342	メッセージ	‘ 関数修飾子 ’ keyword is not allowed
	原因	この関数修飾子は使用することができません。

9.3.5 変換に対するエラー・メッセージ

表 9-5 変換に対するエラー・メッセージ〈0401 〜〉

エラー番号	エラー・メッセージ	
W0401	メッセージ	Conversion may lose significant digits
	原因	long から int への変換などが行われています。値が失われる可能性がありますので，注意してください。
E0402	メッセージ	Incompatible type conversion
	原因	代入文で許されない型変換が行われています。
E0403	メッセージ	Illegal indirection
	原因	整数型の式に * 演算子が使われています。
E0404	メッセージ	Incompatible structure type conversion
	原因	構造体同士，または構造体への代入文で両辺の型が異なっています。
E0405	メッセージ	Illegal lvalue
	原因	左辺値として正しくないものがあります。
E0406	メッセージ	Cannot modify a const object ' 変数名 '
	原因	const 属性の変数の書き換えを行っています。
E0407	メッセージ	Cannot write for read/only sfr 'SFR 名 '
	原因	read only の sfr に対し，書き込みを行っています。
E0408	メッセージ	Cannot read for write/only sfr 'SFR 名 '
	原因	write only の sfr に対し，読み出しを行っています。
E0409	メッセージ	Illegal SFR access 'SFR 名 '
	原因	sft に対して不正なデータの読み出し，または書き込みを行っています。
W0410	メッセージ	Illegal pointer conversion
	原因	ポインタとポインタ以外のものの変換が行われています。
W0411	メッセージ	Illegal pointer combination
	原因	ポインタ同士で異なる型のものを混合して使用しています。
W0412	メッセージ	Illegal pointer combination in conditional expression
	原因	ポインタ同士で異なる型のものを条件式に使用しています。
W0413	メッセージ	Illegal structure pointer combination
	原因	型の異なる構造体へのポインタを混合して使用しています。
E0414	メッセージ	Expected pointer
	原因	ポインタが必要です。
W0415	メッセージ	Conversion may lose significant digits for far pointer
	原因	far ポインタから near ポインタ，int への変換などが行われています。値が失われる可能性がありますので，注意してください。

エラー番号	エラー・メッセージ	
W0416	メッセージ	Illegal type and size (far/near) pointer combination
	原因	ポインタ同士で異なる型，異なるサイズ（far ポインタ／ near ポインタ）のものを混合して使用しています。
W0417	メッセージ	Illegal type and size (far/near) pointer combination in conditional expression
	原因	ポインタ同士で異なる型，異なるサイズ（far ポインタ／ near ポインタ）のものを条件式に使用しています。
W0418	メッセージ	Illegal structure and size (far/near) pointer combination
	原因	構造体へのポインタ同士で，異なる型，異なるサイズ（far ポインタ／ near ポインタ）のものを混合して使用しています。

9.3.6 式に対するエラー・メッセージ

表 9-6 式に対するエラー・メッセージ〈0501 ～〉

エラー番号	エラー・メッセージ	
E0501	メッセージ	Expression syntax
	原因	式の構文エラーが起こりました。
E0502	メッセージ	Compiler limit : too many parentheses
	原因	式中のかっこのネストが 32 を越えました。
W0503	メッセージ	Possible use of ' 変数名 ' before definition
	原因	変数に値が代入される前に，その変数を使用しています。
W0504	メッセージ	Possibly incorrect assignment
	原因	if，while，do 文などで条件式のおもな演算子が代入演算子です。
W0505	メッセージ	Operator ' 演算子 ' has no effect
	原因	演算子にプログラム上の作用がありません。 記述ミスと思われます。
E0507	メッセージ	Expected integral index
	原因	配列の添字に許されるのは整数型の式だけです。
W0508	メッセージ	Too many actual arguments
	原因	関数呼び出しで指定された引数の数が，引数の型のリスト，または関数定義で指定されたパラメータの数より多い状態です。
W0509	メッセージ	Too few actual arguments
	原因	関数呼び出しで指定された引数の数が，引数の型のリスト，または関数定義で指定されたパラメータの数より少ない状態です。
W0510	メッセージ	Pointer mismatch in function ' 関数名 '
	原因	与えられた引数が，引数の型のリストや関数定義で指定されたものとは異なるポインタの型を持ちます。
W0511	メッセージ	Different argument types in function ' 関数名 '
	原因	関数の呼び出しで与えられた引数の型が，引数の型のリストや関数定義と一致していません。
E0512	メッセージ	Cannot call function in norec function
	原因	norec 関数中で関数呼び出しを行っています。 関数呼び出しは，norec 関数中では行うことができません。
E0513	メッセージ	Illegal structure/union member ' メンバ名 '
	原因	構造体の参照で，定義されていないメンバを指しています。
E0514	メッセージ	Expected structure/union pointer
	原因	“->”演算子の前の式が，構造体または共用体へのポインタではなく，構造体，または共用体の名前です。
	ユーザの処置	“->”演算子の前の式を構造体，または共用体へのポインタにしてください。

エラー番号	エラー・メッセージ	
E0515	メッセージ	Expected structure/union name
	原因	“.”演算子の前の式が，構造体または共用体の名前ではなく，構造体，または共用体へのポインタです。
	ユーザの処置	“.”演算子の前の式を構造体，または共用体変数にしてください。
E0516	メッセージ	Zero sized structure ' 構造体名 '
	原因	構造体の大きさが 0 です。
E0517	メッセージ	Illegal structure operation
	原因	構造体に使用できない演算子を使用しています。
E0518	メッセージ	Illegal structure/union comparison
	原因	2 個の構造体，または共用体を比較することができません。
E0519	メッセージ	Illegal bit field operation
	原因	ビット・フィールドに対して許されない記述があります。
E0520	メッセージ	Illegal use of pointer
	原因	ポインタに対して使用できる演算子は，加減，代入，関係，間接（*），メンバ参照（->）だけです。
E0521	メッセージ	Illegal use of floating
	原因	浮動小数点変数に対して，使用することができない演算子が使用されています。
W0522	メッセージ	Ambiguous operators need parentheses
	原因	2 つのシフト，関係，ビット論理演算子が，かっこなしに連続して現れています。
E0523	メッセージ	Illegal bit , boolean type operation
	原因	bit，boolean 型変数に対して許されない演算を行っています。
E0524	メッセージ	'&' on constant
	原因	定数のアドレスは得られません。
E0525	メッセージ	'&' requires lvalue
	原因	‘&’ 演算子は左辺値に代入する式にのみ使用可能です。
E0526	メッセージ	'&' on register variable
	原因	レジスタ変数のアドレスは得ることができません。
E0527	メッセージ	'&' on bit , boolean ignored
	原因	ビット・フィールド，bit，boolean 型変数のアドレスは得ることができません。
W0528	メッセージ	'&' is not allowed array/function , ignored
	原因	配列名や関数名に＆演算子をつける必要はありません。
E0529	メッセージ	Sizeof returns zero
	原因	sizeof 式の値が 0 になっています。

エラー番号	エラー・メッセージ	
E0530	メッセージ	Illegal sizeof operand
	原因	sizeof 式のオペランドは，識別子，または型名でなければなりません。
E0531	メッセージ	Disallowed conversion
	原因	不正なキャストを行っています。
	ユーザの処置	キャストが間違っていないか確かめてください。 定数をポインタにキャストしている場合，メモリ・モデルにより範囲外のアドレスとなる場合もこのエラーになります。
E0532	メッセージ	Pointer on left , needs integral right : ' 演算子 '
	原因	左辺オペランドがポインタであるので，右辺オペランドは整数値でなければなりません。
E0533	メッセージ	Invalid left-or-right operand : ' 演算子 '
	原因	左辺，または右辺オペランドが，演算子に対して不正です。
E0534	メッセージ	Divide check
	原因	/ 演算，% 演算の除数が 0 です。
E0535	メッセージ	Invalid pointer addition
	原因	2 つのポインタを加算してはなりません。
E0536	メッセージ	Must be integral value addition
	原因	ポインタに加算できるものは整数値のみです。
E0537	メッセージ	Illegal pointer subtraction
	原因	ポインタ同士の減算は同じ型でなければなりません。
E0538	メッセージ	Illegal conditional operator
	原因	条件演算子が正しく記述されていません。
E0539	メッセージ	Expected constant expression
	原因	定数式が必要です。
W0540	メッセージ	Constant out of range in comparison
	原因	定数部分式が，もう一方の部分式の型によって許される範囲外の値と比較されています。
E0541	メッセージ	Function argument has void type
	原因	関数の引数が void 型です。
W0543	メッセージ	Undeclared parameter in norec function prototype
	原因	norec 関数のプロトタイプ宣言において，パラメータの宣言がされていません。
E0544	メッセージ	Illegal type for parameter in norec function prototype
	原因	norec 関数のプロトタイプ宣言において，許していない型のパラメータ宣言がされています。
E0546	メッセージ	Too few actual argument for inline function ' 関数名 '
	原因	インライン展開する関数の関数呼び出しで指定された引数の個数が仕様で規定するパラメータの数より少ない状態です。

9.3.7 文に対するエラー・メッセージ

表 9-7 文に対するエラー・メッセージ〈0601 ～〉

エラー番号	エラー・メッセージ	
E0602	メッセージ	Compiler limit : too many characters in logical source line
	原因	論理ソース行の文字数が 2,048 を越えました。
E0603	メッセージ	Compiler limit : too many labels
	原因	ラベル数が 33 を越えました。
E0604	メッセージ	Case not in switch
	原因	case 文が正しい位置に記述されていません。
E0605	メッセージ	Duplicate case ' ラベル名 '
	原因	switch 文の中で同じ case ラベルが 2 度以上記述されています。
E0606	メッセージ	Non constant case expression
	原因	case 文で整数定数以外のものを指定しています。
E0607	メッセージ	Compiler limit : too many case labels
	原因	switch 文の case ラベルが 257 を越えました。
E0608	メッセージ	Default not in switch
	原因	default 文が正しい位置に記述されていません。
E0609	メッセージ	More than one 'default'
	原因	switch 文の中で default 文が複数記述されています。
E0610	メッセージ	Compiler limit : block nest level too depth
	原因	ブロックのネストが 45 を越えました。
E0611	メッセージ	Inappropriate 'else'
	原因	if と else の対応がとれていません。
W0613	メッセージ	Loop entered at top of switch
	原因	switch 文の直後に while，do，for などを指定しています。
W0615	メッセージ	Statement not reached
	原因	絶対に到達しない文があります。
E0617	メッセージ	Do statement must have 'while'
	原因	do の終わりには while が必要です。
E0620	メッセージ	Break/continue error
	原因	break，continue 文の位置が誤っています。
E0621	メッセージ	Void function ' 関数名 'cannot return value
	原因	void 宣言した関数が値を返しています。

エラー番号	エラー・メッセージ	
W0622	メッセージ	No return value
	原因	値を返すべき関数が値を返していません。
	ユーザの処置	値を返す必要がある場合は return 文を追加し，値を返す必要がなければ void 型の関数にしてください。
E0623	メッセージ	No effective code and data , cannot create output file
	原因	有効なコードやデータがないため，出力ファイルを作成することができません。

9.3.8 宣言，関数定義に対するエラー・メッセージ

表 9-8 宣言，関数定義に対するエラー・メッセージ〈0701 ～〉

エラー番号	エラー・メッセージ	
E0701	メッセージ	External definition syntax
	原因	関数が正しく定義されていません。
E0702	メッセージ	Too many callt functions
	原因	callt 関数の宣言が多すぎます。callt 関数は最大 32 個まで宣言することができます。
	ユーザの処置	callt 関数宣言の数を減らしてください。
E0703	メッセージ	Function has illegal storage class
	原因	関数が不正な記憶クラスで指定されています。
E0704	メッセージ	Function returns illegal type
	原因	関数の返り値が不正な型です。
E0705	メッセージ	Too many parameters in norec function
	原因	norec 関数のパラメータが多すぎます。
	ユーザの処置	パラメータを減らしてください。
E0706	メッセージ	Parameter list error
	原因	関数パラメータ・リスト中に誤りがあります。
E0707	メッセージ	Not parameter ' 文字列 '
	原因	関数定義でパラメータでないものを宣言しています。
E0710	メッセージ	Illegal strage class
	原因	関数の外部で auto，register 宣言がおこなわれているか，または関数内で boolean 変数が定義されています。
E0711	メッセージ	Undeclared ' 変数名 ' : function ' 関数名 '
	原因	宣言されていない変数が使用されています。
E0712	メッセージ	Declaration syntax
	原因	宣言文が文法に合っていません。
E0713	メッセージ	Redefined ' シンボル名 '
	原因	同じシンボルが 2 回以上定義されています。
	ユーザの処置	シンボルの定義は 1 回にしてください。
W0714	メッセージ	Too many register variables
	原因	レジスタ変数の宣言が多すぎます。
	ユーザの処置	レジスタ変数を減らしてください。使用可能なレジスタ変数の数については，「CC78K0R C コンパイラ 言語編」のユーザーズ・マニュアルを参照してください。
E0715	メッセージ	Too many sreg variables
	原因	sreg 変数の宣言が多すぎます。

エラー番号	エラー・メッセージ	
E0717	メッセージ	Too many automatic data in norec function
	原因	norec 関数のオートマティック変数が多すぎます。
	ユーザの処置	norec 関数のオートマティック変数を減らしてください。使用可能な変数の数については，「CC78K0R C コンパイラ 言語編」のユーザーズ・マニュアルを参照してください。
E0718	メッセージ	Too many bit , boolean type variables
	原因	bit，boolean 型変数が多すぎます。
	ユーザの処置	bit，boolean，__boolean 型変数を減らしてください。使用可能な変数の数については，「CC78K0R C コンパイラ 言語編」のユーザーズ・マニュアルを参照してください。
E0719	メッセージ	Illegal use of type
	原因	型名が不正に使用されています。
E0720	メッセージ	Illegal void type for ' 識別子 '
	原因	識別子を void で宣言しています。
W0721	メッセージ	Illegal type for register declaration
	原因	register 宣言が，許されない型に指定されています。
	プログラムの処理	register 宣言を無視して処理を継続します。
E0723	メッセージ	Illegal type for parameter in norec function
	原因	norec 関数のパラメータの型が大きすぎます。
E0724	メッセージ	Structure redefinition
	原因	同じ構造体が再定義されています。
W0725	メッセージ	Illegal zero sized structure member
	原因	構造体のメンバとして取られている領域が確保されていません。
E0726	メッセージ	Function cannot be structure/union member
	原因	関数は，構造体，または共用体のメンバであってはなりません。
E0727	メッセージ	Unknown size structure/union ' 名前 '
	原因	サイズが未定義の構造体，または共用体があります。
E0728	メッセージ	Compiler limit : too many structure/union members
	原因	構造体，または共用体のメンバが 256 を越えています。
E0729	メッセージ	Compiler limit : structure/union nesting
	原因	構造体，または共用体のネストが 15 を越えています。
E0730	メッセージ	Bit field outside of structure
	原因	構造体の外でビット・フィールドの宣言が行われています。
E0731	メッセージ	Illegal bit field type
	原因	ビット・フィールドの型に整数型以外の型を指定しています。

エラー番号	エラー・メッセージ	
E0732	メッセージ	Too long bit field size
	原因	ビット・フィールド宣言のビット指定数がその型のビット数を越えています。
E0733	メッセージ	Negative bit field size
	原因	ビット・フィールド宣言のビット指定数が負です。
E0734	メッセージ	Illegal enumeration
	原因	列挙型宣言が文法に合っていません。
E0735	メッセージ	Illegal enumeration constant
	原因	列挙定数が不正です。
E0736	メッセージ	Compiler limit : too many enumeration constants
	原因	列挙定数の数が 255 を越えました。
E0737	メッセージ	Undeclared structure/union/enum tag
	原因	タグが宣言されていません。
E0738	メッセージ	Compiler limit : too many pointer modifying
	原因	ポインタの定義で間接演算子（*）の数が 12 を越えました。
E0739	メッセージ	Expected constant
	原因	配列の宣言で添字に変数を使用しています。
E0740	メッセージ	Negative subscript
	原因	配列の大きさの指定が負です。
E0741	メッセージ	Unknown size array ' 配列名 '
	原因	配列の大きさが不定です。
	ユーザの処置	配列の大きさを指定してください。
E0742	メッセージ	Compiler limit : too many array modifying
	原因	配列の宣言が 12 次元を越えています。
E0743	メッセージ	Array element type cannot be function
	原因	関数の配列は許されません。
W0744	メッセージ	Zero sized array ' 配列名 '
	原因	定義した配列の要素数が 0 です。
W0745	メッセージ	Expected function prototype
	原因	関数プロトタイプ宣言がありません。
E0747	メッセージ	Function prototype mismatch
	原因	関数プロトタイプ宣言に誤りがあります。
	ユーザの処置	関数本体とパラメータ，戻り値の型などが同じか確認してください。
W0748	メッセージ	A function is declared as a parameter
	原因	関数が引数として宣言されています。

エラー番号	エラー・メッセージ	
W0749	メッセージ	Unused parameter ' パラメータ名 '
	原因	パラメータが使用されていません。
E0750	メッセージ	Initializer syntax
	原因	初期化が文法にあっていません。
E0751	メッセージ	Illegal initialization
	原因	初期値設定の定数がその変数の型に合っていません。
W0752	メッセージ	Undeclared initializer name ' 名前 '
	原因	初期化子名が宣言されていません。
E0753	メッセージ	Cannot initialize static with automatic
	原因	オートマチック変数を使って，スタティック変数を初期化することができません。
E0756	メッセージ	Too many initializers ' 配列名 '
	原因	宣言された配列の要素数より初期値の方が大きいです。
E0757	メッセージ	Too many structure initializers
	原因	宣言された構造体のメンバ数より初期値の方が大きいです。
E0758	メッセージ	Cannot initialize a function ' 関数名 '
	原因	関数を初期化することができません。
E0759	メッセージ	Compiler limit : initializers too deeply nested
	原因	初期化要素のネストの深さが制限を越えました。
W0760	メッセージ	Double and long double are treated as IEEE 754 single format
	原因	double，long double は，IEEE 754 の単精度フォーマットで処理します。
W0761	メッセージ	Cannot declare sreg with const or function
	原因	const 宣言されたもの，または関数に，sreg 宣言することができません。
	プログラムの処理	sreg 宣言を無視します。
W0762	メッセージ	Overlapped memory area ' 変数名 1' and ' 変数名 2'
	原因	絶対番地配置指定が行われた変数名 1 と変数名 2 の領域が重複しています。
W0763	メッセージ	Cannot declare const with bit , boolean
	原因	bit，boolean 型変数は，const 宣言することができません。
	プログラムの処理	const 宣言を無視します。
W0764	メッセージ	' 変数名 ' initialized and declared extern-ignored extern
	原因	実体がなく，外部参照している変数を初期化しました。
	プログラムの処理	extern 宣言を無視します。

エラー番号	エラー・メッセージ	
E0765	メッセージ	Undefined static function ' 関数名 '
	原因	同一ファイル内に実体がない static 宣言された関数を参照しました。
E0766	メッセージ	Illegal type for automatic data in norec function
	原因	norec 関数のオートマティック変数の型が大きいです。
E0769	メッセージ	__far is not allowed for callt/interrupt function
	原因	callt ／割り込み関数に __far 修飾子を使用することはできません。
E0770	メッセージ	Parameters are not allowed for interrupt function
	原因	interrupt 関数には引数は許されません。
E0771	メッセージ	Interrupt function must be void type
	原因	interrupt 関数は，void 型でなくてはなりません。
E0772	メッセージ	Callt/norec are not allowed for interrupt function
	原因	interrupt 関数は，callt，norec 宣言を指定することができません。
E0773	メッセージ	Cannot call interrupt function
	原因	interrupt 関数をコールできません。
E0774	メッセージ	Interrupt function can't use with the other kind interrupts
	原因	1 つの interrupt 関数を他の種別の割り込みに使用することはできません。
E0775	メッセージ	Cannot call rtos_task function
	原因	RTOS タスクを呼び出すことはできません。
E0776	メッセージ	Cannot call ret_int/_kernel_int_entry
	原因	ret_int/_kernel_int_entry システム・コールを呼び出すことはできません。
E0777	メッセージ	Cannot allocate rtos_system_call
	原因	RTOS システム・コール関数を配置することはできません。
E0778	メッセージ	Cannot call ext_tsk except in rtos_task
	原因	RTOS タスク関数以外で，ext_tsk システム・コールを呼び出すことはできません。
W0779	メッセージ	Not call ext_tsk in rtos_task
	原因	RTOS タスクにおいて，ext_tsk システム・コールを呼び出していません。
E0780	メッセージ	Zero width for bit field ' メンバ名 '
	原因	ビット・フィールド宣言のビット指定数が 0 のメンバに，メンバ名を指定しています。
W0787	メッセージ	Bit field type is char
	原因	ビット・フィールドの型に char 型が指定されています。

エラー番号	エラー・メッセージ	
E0788	メッセージ	Cannot allocate a __flash function ' 関数名 '
	原因	__flash 関数を配置することはできません。
E0789	メッセージ	'-ZF' option did not specify - cannot allocate an EXT_FUNC function ' 関数名 '
	原因	フラッシュ領域のオブジェクト作成オプション (-zf) が指定されていません。 #pragma EXT_FUNC で指定した関数は配置することができません。
E0790	メッセージ	Callt/__interrupt are not allowed for EXT_FUNC function ' 関数名 '
	原因	#pragma EXT_FUNC で指定した関数には，callt/__interrupt 宣言を指定することはできません。
E0791	メッセージ	'-ZF' option specified - cannot allocate a callt function ' 関数名 '
	原因	フラッシュ領域のオブジェクト作成オプション (-zf) が指定されました。 callt 関数は配置することができません。
E0799	メッセージ	Cannot allocate ' 変数名 ' out of ' アドレス範囲 '
	原因	絶対番地配置指定が行われた変数名に対するアドレス指定が，指定可能なアドレス範囲を越えています。

9.3.9 前処理指令に対するエラー・メッセージ

表 9-9 前処理指令に対するエラー・メッセージ〈0801 ～〉

エラー番号	エラー・メッセージ	
E0801	メッセージ	Undefined control
	原因	# で始まるもので，キーワードとして認識できないものがあります。
E0802	メッセージ	Illegal preprocess directive
	原因	プリプロセス指令に誤りがあります。
	ユーザの処置	プリプロセス指令（#pragma など）がファイルの先頭に記述されているか，または間違いがないか確認してください。
E0803	メッセージ	Unexpected non-whitespace before preprocess directive
	原因	プリプロセス指令の前に空白文字以外の文字があります。
W0804	メッセージ	Unexpected characters following ' プリプロセス指令 ' directive - newline expected
	原因	プリプロセス指令の後に余分な文字があります。
E0805	メッセージ	Misplaced else or elif
	原因	#if，#ifdef，#ifndef と #else，#elif の対応がとれていません。
E0806	メッセージ	Misplaced endif
	原因	#if，#ifdef，#ifndef と #endif の対応がとれていません。
E0807	メッセージ	Compiler limit : too many conditional inclusion nesting
	原因	条件コンパイルのネストが 255 を越えました。
E0810	メッセージ	Cannot find include file ' ファイル名 '
	原因	インクルード・ファイルが見つかりません。
	ユーザの処置	環境変数 INC78K0R にインクルード・ファイルのあるパスを設定するか，-i でパスを設定してください。
E0811	メッセージ	Too long file name ' ファイル名 '
	原因	ファイル名が長すぎます。
E0812	メッセージ	Include directive syntax
	原因	#include 文の定義でファイル名が ” “，または < > で正しく囲まれていません。
E0813	メッセージ	Compiler limit : too many include nesting
	原因	インクルード・ファイルのネストが 8 を越えました。
E0814	メッセージ	Illegal macro name
	原因	マクロ名が正しくありません。
E0815	メッセージ	Compiler limit : too many macro nesting
	原因	マクロのネストが 200 を越えました。
W0816	メッセージ	Redefined macro name ' マクロ名 '
	原因	マクロ名が再定義されています。

エラー番号	エラー・メッセージ	
W0817	メッセージ	Redefined system macro name ' マクロ名 '
	原因	システム・マクロ名が再定義されています。
E0818	メッセージ	Redeclared parameter in macro ' マクロ名 '
	原因	マクロ定義内のパラメータ・リストに同じ識別子が現れています。
W0819	メッセージ	Mismatch number of parameter ' マクロ名 '
	原因	#define で定義したパラメータの数と参照するときのパラメータの数が異なっています。
E0821	メッセージ	Illegal macro parameter ' マクロ名 '
	原因	関数形式マクロで（）内の記述が正しくありません。
E0822	メッセージ	Missing) ' マクロ名 '
	原因	関数形式マクロで #define 定義の同じ行内に“) ”が見つかりません。
E0823	メッセージ	Too long macro expansion ' マクロ名 '
	原因	マクロ展開時の実引数が長すぎます。
W0824	メッセージ	Identifier truncate to ' マクロ名 '
	原因	マクロ名が長すぎます。
	プログラムの処理	表示されている ' マクロ名 ' に短縮します。
W0825	メッセージ	Macro recursion ' マクロ名 '
	原因	#define 定義がリカーシブになっています。
E0826	メッセージ	Compiler limit : too many macro defines
	原因	マクロ定義数が 10000 を越えました。
E0827	メッセージ	Compiler limit : too many macro parameters
	原因	1 つのマクロ定義，呼び出しのパラメータが 31 を越えました。
E0828	メッセージ	Not allowed #undef for system macro name
	原因	システム・マクロ名が #undef により指定されています。
W0829	メッセージ	Unrecognized pragma ' 文字列 '
	原因	この文字列はサポートしていません。
	ユーザの処置	キーワードなどが間違っていないか確かめてください。 #pragma section で間違ったセグメントを指定した場合もこのワーニングになります。
E0830	メッセージ	No chip specifier : #pragma pc ( )
	原因	デバイス種別指定がありません。
E0831	メッセージ	Illegal chip specifier : '#pragma pc (デバイス種別)'
	原因	デバイス種別指定に誤りがあります。
W0832	メッセージ	Duplicated chip specifier
	原因	デバイス種別指定が重複しています。

エラー番号	エラー・メッセージ	
E0833	メッセージ	Expected #asm
	原因	#asm がありません。
E0834	メッセージ	Expected #endasm
	原因	#endasm がありません。
W0835	メッセージ	Too many characters in assembler source line
	原因	アセンブラ・ソースの 1 行が長すぎます。
W0836	メッセージ	Expected assembler source
	原因	#asm と #endasm の間にアセンブラ・ソースがありません。
W0837	メッセージ	Output assembler source file , not object file
	原因	#asm ブロック，または __asm 文があります。 オブジェクト・ファイルの代わりにアセンブラ・ソースを出力します。
	ユーザの処置	#asm，および __asm 文記述をオブジェクト・ファイルに出力するために -a，または -sa オプションを指定し，出力アセンブラ・ファイルをアセンブルしてください。
E0838	メッセージ	Duplicated pragma VECT or INTERRUPT or RTOS_INTERRUPT ' 文字列 '
	原因	#pragma VECT ' 文字列 '，INTERRUPT ' 文字列 '，または RTOS_INTERRUPT ' 文字列 ' が重複しています。
E0839	メッセージ	Unrecognized pragma VECT or INTERRUPT or RTOS_INTERRUPT ' 文字列 '
	原因	認識されない #pragma VECT ' 文字列 '，INTERRUPT ' 文字列 '，または RTOS_INTERRUPT ' 文字列 ' があります。
W0840	メッセージ	Undefined interrupt function ' 関数名 ' - ignored BANK or SP_SWITCH specified
	原因	定義のない割り込み関数に対して，退避先が指定されています。
	プログラムの処理	レジスタ・バンク指定，スタック切り替え指定を無視します。
E0842	メッセージ	Unrecognized pragma SECTION ' 文字列 '
	原因	認識されない #pragma SECTION ' 文字列 ' があります。
E0843	メッセージ	Unspecified start address of ' セクション名 '
	原因	#pragma section の AT の後に正しい開始アドレスが指定されていません。
E0845	メッセージ	Cannot allocate ' セクション名 ' out of ' アドレス範囲 '
	原因	指定された開始アドレスには指定されたセクションは配置できません。
W0846	メッセージ	Rechanged section name ' セクション名 '
	原因	同じセクション名に対し，重複して変更指定しています。
	プログラムの処理	後に指定されたセクション名を有効として処理を続けます。

エラー番号	エラー・メッセージ	
E0847	メッセージ	Different BANK or SP_SWITCH specified on same interrupt function ' 関数名 '
	原因	同名の割り込み関数に対して異なるレジスタ・バンク指定，あるいはスタック切り替え指定が行われました。
W0849	メッセージ	#pragma statement is not portable
	原因	#pragma 文は ANSI 準拠ではありません。
W0850	メッセージ	Asm statement is not portable
	原因	ASM 文は ANSI 準拠ではありません。
W0851	メッセージ	Data aligned in ' 領域名 '
	原因	セグメント領域あるいは構造体タグをデータ・アラインします。領域名は，セグメント名，あるいは構造体タグです。
W0852	メッセージ	Module name truncate to ' モジュール名 '
	原因	指定されたモジュール名が長すぎます。
	プログラムの処理	表示された ' モジュール名 ' に短縮します。
E0853	メッセージ	Unrecognized pragma NAME ' モジュール名 '
	原因	' モジュール名 ' 中に認識できない文字があります。
E0854	メッセージ	Undefined rtos_task ' 文字列 '
	原因	RTOS タスクの実体が定義されていません。
E0855	メッセージ	Cannot assign rtos_interrupt_handler to non-maskable and software interrupt
	原因	RTOS_INTERRUPT ハンドラでは，ノンマスカブル割り込み，およびソフトウエア割り込みを指定することはできません。
W0856	メッセージ	Rechanged module name ' モジュール名 '
	原因	重複してモジュール名を指定しています。
W0857	メッセージ	Section name truncate to ' セクション名 '
	原因	指定されたセクション名が長すぎます。
	プログラムの処理	表示された ' セクション名 ' に短縮します。 セクション名は，8 文字以内にしてください。
E0858	メッセージ	Unrecognized pragma 'pragma 文字列 ' ' 不正文字列 '
	原因	認識されない #pragma 'pragma 文字列 ' ' 不正文字列 ' があります。
E0859	メッセージ	Cannot allocate EXT_TABLE out of 0xc0-0xff80
	原因	フラッシュ領域分岐テーブルの先頭アドレスは 0xc0 - 0xff80 でなければなりません。
E0860	メッセージ	Redefined #pragma EXT_TABLE
	原因	#pragma EXT_TABLE が再定義されています。

エラー番号	エラー・メッセージ	
E0861	メッセージ	No EXT_TABLE specifier
	原因	フラッシュ領域分岐テーブルの先頭アドレス指定がありません。
	プログラムの処理	-zf オプションは，「セルフ書き換え機能を持つフラッシュ・メモリ製品」においてセルフ書き換え機能を使用する場合にのみ指定してください。
E0862	メッセージ	Illegal EXT_FUNC id specifier : out of 0x0-0xff
	原因	#pragma EXT_FUNC で指定するフラッシュ領域中の関数の ID 値は 0x0 - 0xff でなければなりません。
E0863	メッセージ	Redefined #pragma EXT_FUNC name ' 関数名 '
	原因	#pragma EXT_FUNC で指定する関数名が再定義されています。
E0864	メッセージ	Redefined #pragma EXT_FUNC id 'ID 値 '
	原因	#pragma EXT_FUNC で指定する ID 値が再定義されています。
E0865	メッセージ	Out of range - cannot allocate an EXT_FUNC function ' 関数名 '
	原因	フラッシュ領域分岐テーブルのアドレスが範囲を越えました。 #pragma EXT_FUNC で指定した関数は配置することができません。
E0866	メッセージ	#pragma section found after C body. cannot include file containing #pragma section and without C body at the line
	原因	C の本文記述後に #pragma section 構文がありました。 これ以降，#pragma section 構文があり，C の本文（変数や関数の外部参照宣言を含む）のないファイルは，インクルードすることができません。
E0867	メッセージ	#pragma section found after C body. cannot specify #include after #pragma section in this file
	原因	C の本文記述後に #pragma section 構文がありました。 これ以降，# include 文を記述することはできません。
E0868	メッセージ	#include found after C body. cannot specify #pragma section after #include directive
	原因	C の本文記述後に #include 文がありました。 これ以降，#pragma section 構文を記述することはできません。
W0869	メッセージ	' セクション名 ' section cannot change after C body
	原因	指定したセクションは，C の本文記述後に変更することはできません。
W0870	メッセージ	Data aligned before ' 変数名 ' in ' セクション名 '
	原因	' セクション名 ' 中に配置される ' 変数名 ' の前でデータ・アラインします。
W0871	メッセージ	Data aligned after ' 変数名 ' in ' セクション名 '
	原因	' セクション名 ' 中に配置される ' 変数名 ' の後でデータ・アラインします。
E0899	メッセージ	#error で指定された文字列が出力されます。
	原因	#error 文字列が指定されました。

9.3.10 致命的なファイル I/O，許されない OS 上での起動に対するエラー・メッセージ

表 9-10 致命的なファイル I/O，許されない OS 上での起動に対するエラー・メッセージ〈0901 ～〉

エラー番号	エラー・メッセージ	
F0901	メッセージ	File I/O error
	原因	ファイルの入出力の際に物理的な I/O エラーが発生しました。
	ユーザの処置	中間ファイルが原因の場合，コンベンショナル・メモリを増やすか EMS，または XMS メモリを使用してください。
F0902	メッセージ	Cannot open ' ファイル名 '
	原因	ファイルをオープンすることができません。
	ユーザの処置	デバイス・ファイルが通常のサーチ・パスにインストールされているか確認してください。パスは，-y オプションでも指定することができます。「5.4 デバイス・ファイルのサーチ・パス」を参照してください。
F0903	メッセージ	Cannot open overlay file ' ファイル名 '
	原因	オーバレイ・ファイルをオープンすることができません。
F0904	メッセージ	Cannot open temp
	原因	入力用のテンポラリ・ファイルをオープンすることができません。
F0905	メッセージ	Cannot create ' ファイル名 '
	原因	ファイルの create エラーが発生しました。
F0906	メッセージ	Cannot create temp
	原因	出力用のテンポラリ・ファイルの create エラーが発生しました。
	ユーザの処置	環境変数 TMP が設定されているか確認してください。
F0907	メッセージ	No available data block
	原因	ドライブのファイル容量の不足により，テンポラリ・ファイルを作成することができません。
F0908	メッセージ	No available directory space
	原因	ドライブのフォルダ・エリアの不足により，テンポラリ・ファイルを作成することができません。
F0909	メッセージ	R/O : read/only disk
	原因	ドライブが read only 属性のため，テンポラリ・ファイルを作成することができません。
F0910	メッセージ	R/O file : read/only , file opened read/only mode
	原因	次の理由により，テンポラリ・ファイルの write エラーが発生しました。 - テンポラリ・ファイルと同一名のファイルがドライブ上にすでに存在し，read only 属性が与えられています。 - 内部矛盾により，出力テンポラリ・ファイルをリード・オンリー属性でオープンしています。

エラー番号	エラー・メッセージ	
F0911	メッセージ	Reading unwritten data , no available directory space
	原因	次の理由により，出力エラーが発生しました。 - EOF を越えて入力を行おうとしました。 - ドライブのフォルダ・エリアの不足により，テンポラリ・ファイルを作成することができません。
F0912	メッセージ	Write error on temp
	原因	出力用のテンポラリ・ファイルへの write エラーが発生しました。
	ユーザの処置	ソースの式が複雑（ネストが深いなど）が原因の可能性があります。問い合わせ先へ連絡してください。
F0914	メッセージ	Insufficient memory in hostmachine
	原因	メモリ不足のため，CC78K0R を起動することができません。
	ユーザの処置	コンベンショナル・メモリのフリー領域を増やしてください。
W0915	メッセージ	Asm statement found. skip to jump optimize this function' 関数名 '
	原因	#asm ブロック，または __asm 文が検出されました。 この関数はジャンプ最適化を行いません。W0837 の処置を行ってください。
F0922	メッセージ	Heap overflow : please retry compile without -QJ
	原因	ジャンプ最適化でメモリのオーバフローが発生しました。 -qj オプションを指定せずに，再コンパイルする必要があります。
F0923	メッセージ	Illegal device file format
	原因	古いフォーマットのデバイス・ファイルを参照しました。

9.4 PM+ のエラー・メッセージ一覧

表 9-11 PM+ のエラー・メッセージ

エラー種別	エラー・メッセージ	
!	メッセージ	1 行文字数で指定された値が不正です。 指定可能範囲は 72 ≦ LW ≦ 132 です。
	原因	1 行文字数として入力可能範囲外の数値が記述されています。
	ユーザの処置	指定可能範囲の数値で，再度試してみてください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	1 ページ行数で指定された値が不正です。 指定可能範囲は 0，20 ≦ LL ≦ 32,767 です。
	原因	1 ページ行数として入力可能範囲外の数値が記述されています。
	ユーザの処置	指定可能範囲の数値で，再度試してみてください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	タブ文字長で指定された値が不正です。 指定可能範囲は 0 ≦ LT ≦ 8 です。
	原因	タブ文字長として入力可能範囲外の数値が記述されています。
	ユーザの処置	指定可能範囲の数値で，再度試してみてください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	ワーニング・レベルで指定された値が不正です。 指定可能範囲は 0 ≦ W ≦ 2 です。
	原因	ワーニング・レベルとして入力可能範囲外の数値が記述されています。
	ユーザの処置	指定可能範囲の数値で，再度試してみてください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	フォルダが存在しません。 作成しますか？
	原因	存在しないフォルダが記述されています。
	ユーザの処置	フォルダを作成する場合は [OK] ボタンを，フォルダを作成しない場合は [キャンセル] ボタンを押してください。
	ボタン	OK : フォルダを作成し，メッセージを閉じます。 キャンセル : フォルダを作成せずに，メッセージを閉じます。
!	メッセージ	ドライブの準備ができていません。 ドライブを準備してください。
	原因	指定したドライブが見つかりません。
	ユーザの処置	正しいドライブ名を指定してください。
	ボタン	再試行 : 再度アクセスを試みます。 キャンセル : 無視します。

エラー種別	エラー・メッセージ	
!	メッセージ	ファイル名として不適切な文字が含まれています。
	原因	OS，または CC78K0R が使用することのできない文字をファイル名に使用しました。
	ユーザの処置	ファイル名として，OS で使用することのできない文字，および CC78K0R が使用することのできない“#”，“ , ”を使用しないでください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	フォルダの作成に失敗しました。
	原因	ディスク容量不足，書き込み禁止等の理由で，OS からフォルダ作成を拒否されました。
	ユーザの処置	ディスク容量，書き込みの可否を確認してください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	無効なオプションを無視しました。
	原因	プロジェクト・ファイルのオプション情報に，CC78K0R ではワーニング，またはエラーとなるオプションの組み合わせが存在します。
	ユーザの処置	オプションの指定に矛盾がないかを確認してください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	オプション情報を読み込めません。
	原因	オプション情報読込指定で，指定されたファイルに，有効なオプション情報が含まれていません。
	ユーザの処置	指定したオプション情報ファイルが同じシリーズのものか，または破損していないかを確認してください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	パスまたはファイルが見つかりません。 パスおよびファイル名を確認してください。
	原因	インクルード・ファイル・パス等の存在するパス，またはファイル名が必要な指定で，存在しないパス，またはファイルを指定しました。
	ユーザの処置	パス，およびファイル名を確認し，存在するパスを指定してください。
	ボタン	OK : メッセージを閉じます。
×	メッセージ	%s が PATH 環境変数で示されるディレクトリに登録されていません。
	原因	cc78k0r.exe が見つかりません。
	ユーザの処置	CC78K0R のインストールが正しく行われているか，または PATH 環境変数が正しく設定されているかを確認してください。
	ボタン	OK : メッセージを閉じます。
!	メッセージ	インクルード・ファイル・パスで指定したパスが重複しています。
	原因	インクルード・ファイル・パスの指定が重複しています。
	ユーザの処置	インクルード・ファイル・パスが重複しないように指定してください。
	ボタン	OK : メッセージを閉じます。

エラー種別	エラー・メッセージ	
!	メッセージ	インクルード・ファイル・パスで指定したパスの長さが不正です。
	原因	インクルード・ファイル・パスで指定したパスの長さが，範囲を越えています。
	ユーザの処置	正しいパス名を指定してください。
	ボタン	OK　　：メッセージを閉じます。
!	メッセージ	インクルード・ファイル・パスで指定したパスの指定数が不正です。 パスは 64 個まで指定可能です。
	原因	インクルード・ファイル・パスの指定数が，範囲を越えています。
	ユーザの処置	パスの指定数を 64 個以下にしてください。
	ボタン	OK　　：メッセージを閉じます。
!	メッセージ	定義マクロで指定したマクロが重複しています。
	原因	定義マクロの指定が重複しています。
	ユーザの処置	定義マクロが重複しないように指定してください。
	ボタン	OK　　：メッセージを閉じます。
!	メッセージ	定義マクロで指定したマクロの長さが不正です。 マクロは 256 文字まで指定可能です。
	原因	定義マクロで指定した文字の長さが，範囲を越えています。
	ユーザの処置	マクロ名を 256 文字以下にしてください。
	ボタン	OK　　：メッセージを閉じます。
!	メッセージ	定義マクロで指定したマクロの指定数が不正です。 定義マクロと未定義マクロ合わせて 30 個まで指定可能です。
	原因	定義マクロ指定と未定義マクロ指定の数が，範囲を越えています。
	ユーザの処置	定義マクロ指定と未定義マクロ指定を合わせた数を 30 個以下にしてください。
	ボタン	OK　　：メッセージを閉じます。
!	メッセージ	未定義マクロで指定したマクロが重複しています。
	原因	未定義マクロの指定が重複しています。
	ユーザの処置	未定義マクロが重複しないように指定してください。
	ボタン	OK　　：メッセージを閉じます。
!	メッセージ	未定義マクロで指定したマクロの長さが不正です。 未定義マクロは 256 文字まで指定可能です。
	原因	未定義マクロで指定した文字の長さが，範囲を越えています。
	ユーザの処置	マクロ名を 256 文字以下にしてください。
	ボタン	OK　　：メッセージを閉じます。

エラー種別	エラー・メッセージ	
!	メッセージ	未定義マクロで指定したマクロの指定数が不正です。 定義マクロと未定義マクロ合わせて 30 個まで指定可能です。
	原因	未定義マクロ指定と未定義マクロ指定の数が，範囲を越えています。
	ユーザの処置	定義マクロ指定と未定義マクロ指定を合わせた数を 30 個以下にしてください。
	ボタン	OK　　：メッセージを閉じます。
!	メッセージ	変更内容を個別オプションを設定している（ソース名）に反映できないオプションがありました。
	原因	全体オプションを個別オプションに反映するときに，指定内容に矛盾，または範囲を越える指定がありました。
	ユーザの処置	反映できないオプション指定は無視します。 必要に応じて，個別オプションの設定を確認してください。
	ボタン	OK　　：メッセージを閉じます。

付録 A サンプル・プログラム

この章では，CC78K0R のサンプル・プログラムを紹介します。

A.1 C ソース・モジュール・ファイル

```
#define TRUE    1
#define FALSE   0
#define SIZE    200

char    mark [ SIZE + 1 ] ;

main ( )
{
        int     i , prime , k , count ;

        count = 0 ;

        for ( i = 0 ; i <= SIZE ; i++ )
                mark [ i ] = TRUE ;
        for ( i = 0 ; i <= SIZE ; i++ ) {
                if ( mark [ i ] ) {
                        prime = i + i + 3 ;
                        printf ( "%6d" , prime ) ;
                        count++ ;
                        if ( ( count%8 ) == 0 ) putchar ( '\n' ) ;
                        for ( k = i + prime ; k <= SIZE ; k += prime )
                                mark [ k ] = FALSE ;
                }
        }
        printf ( "\n%d primes found." , count ) ;
}

printf ( char *s , int i )
{
        int     j ;
        char    *ss ;

        j = i ;
        ss = s ;
}

putchar ( char c )
{
        char    d ;
        d = c ;
}
```

A.2 実行例

```
C>cc78K0R -cf1166a0 prime.c -a -p -x -e -ng
```

```
78K0R Series C Compiler Vx.xx [ xx xxx xxxx ]
Copyright (C) NEC Electronics Corporation xxxx, xxxx

prime.c ( 18 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 20 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 26 ) : CC78K0R warning W0622 : No return value
prime.c ( 35 ) : CC78K0R warning W0622 : No return value
prime.c ( 41 ) : CC78K0R warning W0622 : No return value

Target chip : uPD78F1166_A0
Device file : Vx.xx

Compilation complete, 0 error(s) and 5 warning(s) found.
```

A.3 出力リスト

A.3.1 アセンブラ・ソース・モジュール・ファイル

```
; 78K0R Series C Compiler Vx.xx Assembler Source
;                                     Date : xx xxx xxxx Time : xx : xx : xx
; Command       : -cf1166a0 prime.c -a -p -x -e -ng
; In-file       : prime.c
; Asm-file      : prime.asm
; Para-file     :

$PROCESSOR ( f1166a0 )
$NODEBUG
$NODEBUGA
$KANJICODE EUC
$TOL_INF        03FH , 100H , 02H , 00H , 00H

        EXTRN   _@RTARG0
        EXTRN   @@isrem
        PUBLIC  _mark
        PUBLIC  _main
        PUBLIC  _printf
        PUBLIC  _putchar

@@CNST  CSEG    MIRRORP
L0011 : DB      '%6d'
        DB      00H
L0017 : DB      ' '
        DB      0AH
        DB      'd primes found.'
        DB      00H

@@DATA  DSEG    BASEP
_mark : DS      ( 201 )
        DS      ( 1 )

; line  5
; line  8

@@CODE  CSEG    BASE
_main :
        push    hl                                      ; [ INF ] 1 , 1
        subw    sp , #08H                               ; [ INF ] 2 , 1
        movw    hl , sp                                 ; [ INF ] 3 , 1
; line  11
        clrw    ax                                      ; [ INF ] 1 , 1
        movw    [ hl ] , a              ; count         ; [ INF ] 1 , 1
; line  13
        movw    [ hl + 6 ] , ax         ; i             ; [ INF ] 2 , 1
L0003 :
        movw    ax , [ hl + 6 ]         ; i             ; [ INF ] 2 , 1
        cmpw    ax , #0C8H              ; 200           ; [ INF ] 3 , 1
        orl     CY , a.7                                ; [ INF ] 2 , 1
        skc                                             ; [ INF ] 2 , 1
        bnz     $L0004                                  ; [ INF ] 2 , 4
; line  14
        movw    ax , [ hl + 6 ]         ; i             ; [ INF ] 2 , 1
        movw    bc , ax                                 ; [ INF ] 1 , 1
        mov     _mark [ bc ] , #01H     ; 1             ; [ INF ] 4 , 1
```

```
        movw    ax , [ hl + 6 ]         ; i             ; [ INF ] 2 , 1
        incw    ax                                      ; [ INF ] 1 , 1
        movw    [ hl + 6 ] , ax         ; i             ; [ INF ] 2 , 1
        br      $L0003                                  ; [ INF ] 2 , 4
L0004 :
; line  15
        clrw    ax                                      ; [ INF ] 1 , 1
        movw    [ hl + 6 ] , ax         ; i             ; [ INF ] 2 , 1
L0006 :
        movw    ax , [ hl + 6 ]         ; i             ; [ INF ] 2 , 1
        cmpw    ax , #0C8H              ; 200           ; [ INF ] 3 , 1
        or1     CY , a.7                                ; [ INF ] 2 , 1
        skc                                             ; [ INF ] 2 , 1
        bnz     !L0007                                  ; [ INF ] 2 , 4
; line  16
        movw    ax , [ hl + 6 ]         ; i             ; [ INF ] 2 , 1
        addw    ax , #loww ( _mark )                    ; [ INF ] 3 , 1
        movw    de , ax                                 ; [ INF ] 1 , 1
        mov     a , [ de ]                              ; [ INF ] 2 , 2
        cmp0    a                                       ; [ INF ] 1 , 1
        bz      $L0015                                  ; [ INF ] 2 , 4
; line  17
        movw    ax , [ hl + 6 ]         ; i             ; [ INF ] 2 , 1
        addw    ax , ax                                 ; [ INF ] 1 , 1
        addw    ax , #03H               ; 3             ; [ INF ] 3 , 1
        movw    [ hl + 4 ] , ax         ; prime         ; [ INF ] 2 , 1
; line  18
        push    ax                                      ; [ INF ] 1 , 1
        movw    ax , #loww ( L0011 )    ; 0             ; [ INF ] 3 , 1
        call    !!_printf                               ; [ INF ] 4 , 3
        pop     ax                                      ; [ INF ] 1 , 1
; line  19
        movw    ax , [ hl ]             ; count         ; [ INF ] 1 , 1
        incw    ax                                      ; [ INF ] 1 , 1
        movw    [ hl ] , ax             ; count         ; [ INF ] 1 , 1
; line  20
        movw    _@RTARG0 , ax                           ; [ INF ] 2 , 1
        movw    ax , #08H               ; 8             ; [ INF ] 3 , 1
        call    !!@@isrem                               ; [ INF ] 3 , 3
        or      a , x                                   ; [ INF ] 2 , 1
        bnz     $L0012                                  ; [ INF ] 2 , 4
        movw    ax , #0AH               ; 10            ; [ INF ] 3 , 1
        call    !!_putchar                              ; [ INF ] 4 , 3
L0012 :
; line  21
        movw    ax ,  [ hl + 6 ]        ; i             ; [ INF ] 2 , 1
        xchw    ax , bc                                 ; [ INF ] 1 , 1
        movw    ax , [ hl + 4 ]         ; prime         ; [ INF ] 2 , 1
        addw    ax , bc                                 ; [ INF ] 1 , 1
        movw    [ hl + 2 ] , ax         ; k             ; [ INF ] 2 , 1
L0014 :
        movw    ax , [ hl + 2 ]         ; k             ; [ INF ] 2 , 1
        cmpw    ax , #0C8H              ; 200           ; [ INF ] 3 , 1
        or1     CY , a.7                                ; [ INF ] 2 , 1
        skc                                             ; [ INF ] 2 , 1
        bnz     $L0015                                  ; [ INF ] 2 , 4
; line  22
        movw    ax , [ hl + 2 ]         ; k             ; [ INF ] 2 , 1
        movw    bc , ax                                 ; [ INF ] 1 , 1
        mov     _mark [ bc ] , #00H     ; 0             ; [ INF ] 4 , 1
        movw    ax , [ hl + 2 ]         ; k             ; [ INF ] 2 , 1
```

```
        xchw    ax , bc                                 ; [ INF ] 1 , 1
        movw    ax , [ hl + 4 ]         ; prime         ; [ INF ] 2 , 1
        addw    ax , bc                                 ; [ INF ] 1 , 1
        movw    [ hl + 2 ] , ax         ; k             ; [ INF ] 2 , 1
        br      $L0014                                  ; [ INF ] 2 , 4
L0015 :
; line  24
        movw    ax , [ hl + 6 ]         ; i             ; [ INF ] 2 , 1
        incw    ax                                      ; [ INF ] 1 , 1
        movw    [ hl + 6 ] , ax         ; i             ; [ INF ] 2 , 1
        br      $L0006                                  ; [ INF ] 2 , 4
L0007 :
; line  25
        movw    ax , [ hl ]             ; count         ; [ INF ] 1 , 1
        push    ax                                      ; [ INF ] 1 , 1
        movw    ax , #loww ( L0017 )    ; 0             ; [ INF ] 3 , 1
        call    !!_printf                               ; [ INF ] 4 , 3
        pop     ax                                      ; [ INF ] 1 , 1
; line  26
        addw    sp , #08H                               ; [ INF ] 2 , 1
        pop     hl                                      ; [ INF ] 1 , 1
        ret                                             ; [ INF ] 1 , 6
; line  29
_printf :
        push    hl                                      ; [ INF ] 1 , 1
        push    ax                                      ; [ INF ] 1 , 1
        subw    sp , #04H                               ; [ INF ] 2 , 1
        movw    hl , sp                                 ; [ INF ] 3 , 1
; line  33
        movw    ax , [ hl + 12 ]        ; i             ; [ INF ] 2 , 1
        movw    [ hl + 2 ] , ax         ; j             ; [ INF ] 2 , 1
; line  34
        movw    ax , [ hl + 4 ]         ; s             ; [ INF ] 2 , 1
        movw    [ hl ] , ax             ; ss            ; [ INF ] 1 , 1
; line  35
        addw    sp , #04H                               ; [ INF ] 2 , 1
        pop     hl                                      ; [ INF ] 1 , 1
        ret                                             ; [ INF ] 1 , 6
; line  38
_putchar :
        push    hl                                      ; [ INF ] 1 , 1
        movw    hl , ax                                 ; [ INF ] 1 , 1
; line  40
        mov     a , l                                   ; [ INF ] 1 , 1
        mov     h , a                                   ; [ INF ] 1 , 1
; line  41
        pop     hl                                      ; [ INF ] 1 , 1
        ret                                             ; [ INF ] 1 , 6
        END


; *** Code Information ***
;
; $FILE /auto/proj/cmp/cc.new/work/egashira/cc78sk0r/src/test/prime2.c
;
; $FUNC main ( 8 )
;       bc = ( void )
;       CODE SIZE = 155 bytes , CLOCK_SIZE = 117 clocks , STACK_SIZE = 16 bytes
;
; $CALL printf ( 18 )
;       bc = ( pointer : ax , int : [ sp + 2 ] )
```

```
;
; $CALL putchar ( 20 )
;       bc = ( int : ax )
;
; $CALL printf ( 25 )
;       bc = ( pointer : ax , int : [ sp + 2 ] )
;
; $FUNC printf ( 29 )
;       bc = ( pointer s : ax , int i : [ sp + 4 ] )
;       CODE SIZE = 18 bytes , CLOCK_SIZE = 16 clocks , STACK_SIZE = 8 bytes
;
; $FUNC putchar ( 38 )
;       bc = ( char c : x )
;       CODE SIZE = 6 bytes , CLOCK_SIZE = 11 clocks , STACK_SIZE = 2 bytes

; Target chip : uPD78F1166_A0
; Device file : Vx.xx
```

A.3.2 プリプロセス・リスト・ファイル

```
/*
78K0R Series C Compiler Vx.xx Preprocess List     Date : xx xxx xxxx Page : 1
Command         : -cf1166a0 prime.c -a -p -x -e -ng
In-file         : prime.c
PPL-file        : prime.ppl
Para-file       :
*/

        1 :   #define TRUE    1
        2 :   #define FALSE   0
        3 :   #define SIZE    200
        4 :
        5 :   __far char  mark [ SIZE + 1 ] ;
        6 :
        7 :   main ( )
        8 :   {
        9 :       int i , prime , k , count ;
        10 :
        11 :      count = 0 ;
        12 :
        13 :      for ( i = 0 ; i <= SIZE ; i++ )
        14 :          mark [ i ] = TRUE ;
        15 :      for ( i  = 0 ; i <= SIZE ; i++ ) {
        16 :          if ( mark [ i ] ) {
        17 :              prime = i + i + 3 ;
        18 :              printf ( "%6d", prime ) ;
        19 :              count++ ;
        20 :              if ( ( count%8 ) == 0 ) putchar ( '\n' ) ;
        21 :                  for ( k = i + prime ; k <= SIZE ; k += prime )
        22 :                      mark [ k ] = FALSE ;
        23 :              }
        24 :      }
        25 :      printf ( "\n%d primes found." , count ) ;
        26 :  }
        27 :
        28 :  printf ( char *s , int i )
        29 :  {
        30 :      int     j ;
        31 :      char    *ss ;
        32 :
        33 :      j = i ;
        34 :      ss = s ;
        35 :  }
        36 :
        37 :  putchar ( char c )
        38 :  {
        39 :      char    d ;
        40 :      d = c ;
        41 :  }


/*
Target chip : uPD78F1166_A0
Device file : Vx.xx
*/
```

A.3.3 クロスリファレンス・リスト・ファイル

```
78K0R Series C Compiler Vx.xx Cross reference List  Date : XX XXX XXXX Page : 1

Command     : -cf1166a0 prime.c -a -p -x -e –ng
In-file     : prime.c
Xref-file   : prime.xrf
Para-file   :

ATTRIB  MODIFY  TYPE    SYMBOL  DEFINE  REFERENCE

EXTERN  FAR     array   mark     5      14  16  22
EXTERN  FAR     func    main     7
AUTO1           int     i        9      13  13  13  14  15  15  15  16  17  17
                                        21
AUTO1           int     prime    9      17  18  21  21
AUTO1           int     k        9      21  21  21  22
AUTO1           int     count    9      11  19  20  25
EXTERN  FAR     func    printf  28      18  25
EXTERN  FAR     func    putchar 37      20
PARAM           pointer s       28      34
PARAM           int     i       28      33
AUTO1           int     j       30      33
AUTO1           pointer ss      31      34
REG1            char    c       37      40
PARAM
REG1            char    d       39      40
                #define TRUE     1      14
                #define FALSE    2      22
                #define SIZE     3       5  13  15  21


 Target chip : uPD78F1166_A0
 Device file : Vx.xx
```

A.3.4 エラー・リスト・ファイル

```
prime.c ( 18 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 20 ) : CC78K0R warning W0745 : Expected function prototype
prime.c ( 26 ) : CC78K0R warning W0622 : No return value
prime.c ( 35 ) : CC78K0R warning W0622 : No return value
prime.c ( 41 ) : CC78K0R warning W0622 : No return value


Target chip : uPD78F1166_A0
Device file : Vx.xx
Compilation complete, 0 error(s) and 5 warning(s) found.
```

付録 B 使用上の注意事項

この章では，CC78K0R を使用する際の注意事項を示します。

表 B-1 使用上の注意事項

項番	注意事項
(1)	**【オプション指定】** - 複数指定が可能でないオプションを複数指定した場合には，後に指定した方を優先します。 - -c に続けて指定する種別を省略することはできません。省略した場合は，致命的エラーとなります。 -c オプションで指定しない場合は，C ソース・モジュール・ファイル中に #pragma pc (種別) を記述してください。 また，実行の際のオプションと C ソース中の種別が異なった場合には，オプションの指定を優先します。このとき，ワーニング・メッセージが出力されます。 - ヘルプ指定があった場合には，他のすべてのオプションは無効になります。
(2)	**【ファイルの出力先】** オブジェクト・モジュール・ファイルの出力先は，ディスク型ファイルのみです。
(3)	**【エラー・メッセージ】** ファイルに文法的誤りがあった場合には，エラー・メッセージにファイル名が付加されます。また，デバイス型ファイルを指定禁止箇所で指定した場合は，指定文字列がそのまま出力されます。それ以外のときは，ドライブ名とパス名と拡張子が必ず付加されます。
(4)	**【ソース・ファイル名】** CC78K0R では，ソース・ファイル名の拡張子を除いた部分（プライマリ名）を，デフォルトでモジュール名として使用します。そのため，使用可能なソース・ファイル名には，若干の制限があります。 - ファイル名の長さは，OS の許す範囲内のプライマリ名と拡張子で構成し，プライマリ名と拡張子の間は，ドット（ . ）で区切る形式としてください。 - プライマリ名，拡張子ともに，使用可能な文字は，OS が許している文字から，括弧（ ( ) ），セミコロン（ ; ），コンマ（ , ）を除いた文字とします。ただし，ファイル名とパス名の先頭に，ハイフン（ - ）を使用することはできません。また，スペースや 2 バイト文字を含むファイル名を指定しないでください。 - パラメータ・ファイル内では，ファイル名とパス名にシャープ（#）を使用することはできません。
(5)	**【インクルード・ファイル】** インクルード・ファイル内で，関数を定義し（宣言を除きます），C ソース中で展開することはできません。 インクルード・ファイル内で定義を行うと，ソース・デバッグ時に正しく定義行が表示されないなどの弊害があります。

項番	注意事項
(6)	**【アセンブラ・ソースを出力して使用する場合】** C ソース・プログラム中に，#asm ブロック，または __asm 文などのアセンブリ言語による記述がある場合，ロード・モジュール・ファイル作成手順は，コンパイル，アセンブル，リンクの順になります。 アセンブリ言語による記述がある場合などのように，CC78K0R で直接オブジェクトを出力せずに，いったんアセンブラ・ソースを出力し，アセンブルして使用する場合には，次の点に注意してください。 - #asm ブロック（#asm から #endasm で囲まれた部分），および __asm 文中で，シンボルを定義する必要があるときには，?L@ の文字列で始まる 8 文字以内のシンボル（たとえば，?L@01，?L@sym など）を使用してください。ただし，このシンボルを外部定義（PUBLIC 宣言）しないでください。また，#asm ブロック，および __asm 文中で，セグメントを定義することはできません。?L@ の文字列で始まるシンボルを使用しない場合，アセンブル時に致命的エラー（F2114）が出力されます。 - C ソースで extern されている変数を #asm ブロック内で使用している場合，他の C 記述部分で参照がないと EXTRN が生成されず，リンク・エラーとなるため，C で参照されない場合は，#asm ブロック内で EXTRN してください。 - C ソース中に，#asm ブロック，および __asm 文がある場合は，アセンブリ記述を有効にするために -a，または -sa オプションを指定して，出力されたアセンブラ・ソースをアセンブルしてください。 PM+ 使用時は，アセンブラ・ソース・ファイルしか出力しないソースに対し，個別オプション指定で -a/-sa オプションを指定するか，全体オプション設定で -a/-sa オプションを指定してください。 - PM+ 使用時に，コンパイラ・オプション -a，または -sa を指定した場合は，コンパイラ・オプション -o/-no にかかわらず RA78K0R を起動します。 - #pragma section 指令でセグメント名を変更する場合，ソース・ファイル名のプライマリ名と同名のセグメント名を指定しないでください。アセンブル時にエラー（F2106）が出力されます。

項番	注意事項

(7)	**【リンク・ディレクティブ・ファイルの作成について】**
	CC78K0R で作成したオブジェクトをリンクする際に，ターゲット・デバイスの ROM/RAM 領域以外の領域を使用する場合，または任意のアドレスに指定してコードやデータを配置させたい場合は，リンク・ディレクティブ・ファイルを作成し，リンク時にリンカ・オプション -d で指定してください。
	リンク・ディレクティブ・ファイルの作成方法については，「RA78K0R アセンブラ・パッケージ 操作編」のユーザーズ・マニュアル，および「CC78K0R に添付されている lk78k0r.dr（smp78k0r フォルダ以下）」を参照してください。
	＜例：ある C ソース・ファイルの初期値なし外部変数（sreg 変数を除く）を外部メモリに配置させたい場合＞
	1. C ソースの先頭で初期値なし外部変数用セクション名を変更する
	`#pragma section @@DATAL EXTDATA`
	`:`
	注意 変更されたセグメントの初期化，および ROM 化はスタートアップ・ルーチンを変更して行うようにしてください。
	2. リンク・ディレクティブ・ファイルを作成する
	＜ lk78k0r.dr ＞
	`memory EXTRAM : ( 040000H , 1000H )`
	`merge EXTDATA : = EXTRAM`
	リンク・ディレクティブ・ファイル作成時には，次の点に注意してください。
	- リンク時に，スタック解決用シンボル生成指定オプション -s を使用する場合には，スタック領域をリンク・ディレクティブ・ファイルの memory ディレクティブで確保し，確保したスタック領域名を，明示的に指定することをお勧めします。領域名を省略した場合は，スタック領域として RAM 領域内（SFR 領域以外）が使用されます。
	＜例：リンク・ディレクティブ・ファイル lk78k0r.dr に追加した場合＞
	`memory EXTRAM : ( 040000H , 1000H )`
	`memory STK : ( 0FB000H , 100H )`
	`merge EXTDATA : = EXTRAM`
	（コマンド・ライン）
	`> lk78k0r s0rml.rel prime.rel -bcl0rm.lib -sSTK -dlk78k0r.dr`
	- 定義しているメモリ領域でリンクすると次のようなリンク・エラーが出力されることがあります。
	`"RA78K0R error E3206 : Segment 'xxx' can't allocate to memory-ignored."`
	【原因】
	定義しているメモリ領域では，領域不足のために，指摘されたセグメントを配置することができないためです。
	【対処方法】
	対処方法は，大きく分けて次の 3 つの手順になります。
	(1) 配置できないセグメントのサイズを調べる（.map ファイル参照）。
	(2) 手順 (1) で調べたセグメントのサイズをもとに，ディレクティブ・ファイルでセグメントが配置されている領域のサイズを拡大する。
	(3) ディレクティブ・ファイル指定オプション -d を指定してリンクする。
	ただし，手順 (1) ではエラーで指摘されたセグメントの種類により，次のようにセグメントのサイズを調べる方法が異なります。
	- コンパイル時に自動生成されるセグメントのとき
	リンクし作成されたマップ・ファイルによりセグメントのサイズを調べる。
	- ユーザが作成したセグメントのとき
	アセンブル・リスト・ファイル（.prn）により配置されなかったセグメントのサイズを調べる。

項番	注意事項
(8)	**【va_start マクロ使用時】** 関数により第一引数のオフセットが異なるため，stdarg.h に定義されている va_start マクロの動作は保証されません。 以下のようにマクロを使い分けてください。 - 第一引数を指定する場合は，va_starttop マクロを使用してください。 - 第二引数以降を指定する場合は，va_start マクロを使用してください。
(9)	**【スタートアップ・ルーチン，ライブラリについて】** - スタートアップ・ルーチン，ライブラリは，使用している実行形式ファイル（cc78k0r.exe）と同じバージョンで提供されているものを使用してください。 - 浮動小数点対応 sprintf，vprintf，vsprintf において，"%f"，"%e"，"%E"，"%g"，"%G" 指定の変換結果の精度以下の値を切り捨ててしまいます。また，"%g"，"%G" 指定の変換結果が精度以上であっても "%f" 変換してしまいます。 浮動小数点対応 sscanf，scanf において，"%f"，"%e"，"%E"，"%g"，"%G" 指定時に 1 文字も有効な文字を読み込まなかった場合 +0 を変換結果とし，" ± " だけの場合± 0 を変換結果とします。 **【回避策】** ありません。

項番	注意事項

(10)	**【ROM 化を行う場合について】**
	ROM 化とは，初期値あり外部変数などの初期値を ROM に配置しておき，システム実行時に RAM にコピーする処理です。CC78K0R では，デフォルトで ROM 化用にコードを生成します。したがって，リンク時に ROM 化処理を含むスタートアップ・ルーチンとリンクする必要があります。
	スモール・モデル，ミディアム・モデル用スタートアップ・ルーチンは，far 領域の ROM 化処理を含んでいません。__far 修飾子などで，変数を far 領域に配置した場合は，コンパクト・モデル，ラージ・モデル用のスタートアップ・ルーチンを使用してください。
	CC78K0R が提供するスタートアップ・ルーチンには次のものがあり，すべて ROM 化処理を含んでいます。
	フラッシュ・メモリのセルフ書き換えモードを使用する場合は，「8.3.1 (3) スタートアップ・ルーチンの使い分け」参照してください。
	スタートアップ・ルーチン：
	- C 標準ライブラリ用の領域を使用しない場合　：　s0rm.rel，s0rl.rel
	- C 標準ライブラリ用の領域を使用する場合　：　s0rml.rel，s0rll.rel
	
	＜使用例＞
	`C>lk78k0r s0rl.rel sample.rel -s -bcl0rxm.lib -bcl0rm.lib -osample.lmf`
	
	`sample.rel` ：ユーザ・プログラムのオブジェクト・モジュール・ファイル
	`s0rl.rel` ：スタートアップ・ルーチン
	`cl0rxm.lib` ：乗算器使用ライブラリ
	`cl0rm.lib` ：ランタイム・ライブラリ，標準ライブラリ
	-s オプションは，スタック・シンボル（_@STBEG，_@STEND）自動生成オプションです。
	
	注意 1　必ずスタートアップ・ルーチンを最初にリンクしてください。
	注意 2　ユーザがライブラリを作成する場合は，CC78K0R が提供するライブラリとは分けて作成し，リンク時に CC78K0R のライブラリよりも前に指定するようにしてください。
	注意 3　CC78K0R のライブラリに，ユーザ関数を追加しないでください。
	注意 4　浮動小数点ライブラリ（cl0r*f.lib）を使用する場合は，通常のライブラリ（cl0r*.lib）と両方リンクする必要があります。
	
	- 浮動小数点対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf を使用する場合
	＜例＞
	`-bmylib.lib -bcl0rmf.lib -bcl0rm.lib`
	- 浮動小数点未対応の sprintf，sscanf，printf，scanf，vprintf，vsprintf を使用する場合
	＜例＞
	`-bmylib.lib -bcl0rm.lib -bcl0rmf.lib`

(11)	**【スタック解決用シンボル生成指定（-s）オプションについて】** CC78K0R では，ユーザはスタック領域を確保することができません。スタック領域を確保するためには，リンク時に -s オプションを指定してください。 PM+ 使用時は，ソース・ファイル指定に C ソースが含まれる場合は，-s オプションが自動的に付加されます。

項番	注意事項
(12)	**【ROM コードについて】** ROM コード発注をする場合には，オブジェクト・コンバータ・オプション -r，および -u を指定してください（指定を解除しないでください）。 ＜例＞ `-r -u0FFH` `-r` : HEX ファイルの内容を，アドレス順にソートします。 `-u 充てん値` : 指定された充てん値を ROM コードの空き領域に埋め込みます。
(13)	**【ヘルプ指定オプションについて】** PM+ で，オプションの説明を表示させるコンパイラ・オプション --/-?/-h は無視されます。ヘルプについては，各ツールの [オプションの設定] ダイアログ内の [ヘルプ] ボタンで参照してください。
(14)	**【-ll オプション指定について】** PM+ 使用時は，-ll オプションに指定可能な最大の数字は，32,767 とします。32,768 以上を指定する場合は，その他のオプションで -ll を指定してください。

項番	注意事項

(15)	**【PM+ 使用時】**
	(a) ユーザが作成したパラメータ・ファイル
	ユーザが作成したパラメータ・ファイルを PM+ に指定すると，PM+ が生成するパラメータ・ファイルにその内容が取りこまれます。パラメータ・ファイルを作成する際には，次のことに注意してください。ビルド時にエラーとなります。
	- PM+ が生成するパラメータ・ファイルと同名のファイルを指定しないでください。
	- デバイス種別指定オプション -c，デバイス・ファイル・サーチ・パス指定オプション -y，およびソース・ファイルは記述しないでください。
	- ユーザが作成したパラメータ・ファイルに記述されたオプションは，正当性のチェックは行われません。
	(b) [アセンブラオプションの設定] ダイアログ
	-c，-f，-y オプション，およびソース・ファイルを指定しないでください。ビルド時にエラーとなります。
	[アセンブラオプションの設定] ダイアログで指定したオプションについては，正当性のチェックを行わないため，記述に誤りがある場合はビルド時にエラーとなります。
	(c) インクルード・ファイルの依存関係
	PM+ でメイク・ファイル作成時に行われるインクルード・ファイルの依存関係のチェックにおいて，#if などの条件文を無視してしまいます。そのため，ビルドに不要なインクルード・ファイルを，必要なファイルであると誤認してしまいます。コメントや文字列として記述された場合は，依存関係がないことが正しく判断されます。
	＜例＞
	`#if 0`
	`#include "header1.h" /* 依存関係ありと判断されてしまう */`
	`#else /* ! zero */`
	`#include "header2.h"`
	`#endif`
	`/*`
	`#include "header3.h"`
	`*/`
	header1.h は，依存関係のチェックにおいて，ビルドに必要であると判断されてしまいます。header1.h ファイルが存在する場合には，PM+ のプロジェクト・ウインドウに header1.h が登録されてしまいます。
	【回避策】
	ありません。ただし，ビルドの動作には影響ありません。
	(d) プロジェクト関連ファイルの設定
	CC78K0R 付属のスタートアップ・ルーチン，標準ライブラリは，PM+ の [プロジェクト] メニュー，およびプロジェクト・ウインドウの右クリックメニュー“プロジェクト関連ファイルの追加”から追加，削除することはできません。
	CC78K0R 付属のスタートアップ・ルーチン，標準ライブラリの設定は，[コンパイラオプションの設定] ダイアログの [スタートアップ・ルーチン] タブで行ってください。

(16)	**【プロトタイプ宣言】**
	関数プロトタイプ宣言において，関数の型指定がない場合，エラー（E0301，E0701）となります。
	＜例＞
	`f ( void ) ; /* E0301 : Syntax error */`
	`/* E0701 : External definition syntax */`
	【回避策】
	関数の型を記述してください。
	＜例＞
	`int f ( void ) ;`

項番	注意事項

(17)	**【エラー・メッセージ出力】**
	関数外で，行頭のキーワードにスペル・ミスがある場合，エラー行の表示位置がずれたり，不適当なエラーを出す場合があります。
	＜例＞
	`extren int  i ;  /* extern のスペルミス。ここでエラーにならない。*/`
	`/* comment */`
	`void  f ( void ) ;`
	`[ EOF ]  /* E0712 などのエラー */`
	【回避策】
	ありません。

(18)	**【前処理指令中のコメント記述】**
	前処理指令の記述において，前処理指令の前や途中，関数形式マクロの並びにコメントを記述すると，エラー（E0803，E0814，E0821 など）となります。
	＜例＞
	`/* com1 */  #pragma  sfr  /* E0803 */`
	`/* com2 */  #define  ONE  1  /* E0803 */`
	`#define  /* com3 */  TWO  2  /* E0814 */`
	`#ifdef  /* com4 */  ANSI_C  /* E0814 */`
	`/* com5 */  #endif`
	`#define  SUB ( p1 , /* com6 */ p2 ) p2 = p1  /* E0821 */`
	【回避策】
	前処理指令の後にコメントを記述してください。
	＜例＞
	`#pragma  sfr  /* com1 */`
	`#define  ONE  1  /* com2 */`
	`#define  TWO  2  /* com3 */`
	`#ifdef  ANSI_C  /* com4 */`
	`#endif  /* com5 */`
	`#define  SUB ( p1 , p2 ) p2 = p1  /* com6 */`

<table>
<tr><th>項番</th><th>注意事項</th></tr>
<tr><td>(19)</td><td>

【構造体 / 共用体 /enum のタグ使用】

関数プロトタイプ宣言で，(構造体，共用体，enum の) タグを定義する前に使用すると，（a）の条件を満たす場合はワーニング，（b）の条件を満たす場合はエラーとなります。

(a) 引数宣言で，そのタグを使用して，構造体，共用体へのポインタを定義すると，関数の呼び出し時にワーニング（W0510）となります。

＜例＞

```
  void    func ( int , struct st ) ;

          struct  st {
                  char    memb1 ;
                  char    memb2 ;
          } st [ ] = {
                  { 1 , ' a ' } , { 2 , ' b ' }
          } ;

  void    caller ( void ) {
                          /* W0510 Pointer mismatch */
          func ( sizeof ( st ) / sizeof ( st [ 0 ] ) , st ) ;
  }
```

(b) 返り値型宣言と引数宣言で，そのタグを使用して，構造体，共用体，enum 型を指定すると，エラー（E0737）となります。

＜例＞

```
                      /* E0737 Undeclared structure/union/enum tag */
  void    func1 ( int , struct st ) ;
                      /* E0737 Undeclared structure/union/enum tag */
  struct  st      func2 ( int ) ;
  struct  st {
          char    memb1 ;
          char    memb2 ;
  } ;
```

【回避策】

構造体，共用体，enum のタグの定義を先に行ってください。

</td></tr>
<tr><td>(20)</td><td>

【関数内の配列 / 構造体 / 共用体の初期化】

関数内で，静的変数のアドレス，定数，文字列以外を用いた，配列 / 構造体 / 共用体の初期化を行うことができません。

＜例＞

```
  void    f ( void ) ;
  void    f ( void ) {
          char    *p , *p1 , *p2 ;
          char    *ca [ 3 ] = { p , p1 , p2 } ;   /* エラー ( E0750 ) */
  }
```

【回避策】

代入文を記述して，初期化の代わりとしてください。

＜例＞

```
  void    f ( void ) ;
  void    f ( void ) {
          char    *ca [ 3 ] ;
          char    *p , *p1 , *p2 ;
          ca [ 0 ] = p ;  ca [ 1 ] = p1 ; ca [ 2 ] = p2 ;
  }
```

</td></tr>
</table>

項番	注意事項

(21)	**【extern callt 関数】**
	extern callt 関数のアドレスを関数テーブルの初期化などで参照し，同じモジュールで callt 関数呼び出しする場合，アセンブル・リストが不正となり，アセンブル時にエラーとなります。
	＜例＞
	`callt extern void funca ( void ) ;`
	`callt extern void funcb ( void ) ;`
	`callt extern void funcc ( void ) ;`
	
	`static void ( * const func [ ] ) ( ) = {`
	`funca , funcb , funcc`
	`} ;`
	`void func2 ( void ) {`
	`funcc ( ) ;`
	`funcb ( ) ;`
	`funca ( ) ;`
	`}`
	【回避策】
	関数テーブルと関数呼び出しのモジュールを分けてください。

(22)	**【構造体を返す関数】**
	関数が構造体そのものを返す場合，返り値を返す処理中に割り込みが発生し，割り込み処理中に同じ関数の呼び出しがあると，割り込み処理終了後に返り値が不正となります。
	＜例＞
	`struct str {`
	`char c ;`
	`int i ;`
	`long l ;`
	`} st ;`
	
	`struct str func ( ) {`
	`/* 割り込み発生 */`
	`:`
	`}`
	
	`void main ( ) {`
	`st = func ( ) ; /* 割り込み発生 */`
	`}`
	上記の処理中，割り込み先で func 関数が呼ばれた場合，st が破壊される可能性があります。
	【回避策】
	ありません。

項番	注意事項

(23)	**【共用体の初期化】**
	構造体，共用体，配列をメンバに持つ共用体の初期化において，初期化子並びを入れ子で指定すると，エラー（E0750）となります。
	＜例＞
	`struct Ss {`
	`  int d1 , d2 ;`
	`} ;`
	`union Au {`
	`  struct Ss t1 ;`
	`} u = { { 1 , 2 } } ;  /* E0750 Initializer syntax */`
	【回避策】
	共用体の初期化子を入れ子で指定しないでください。
	＜例＞
	`struct Ss {`
	`  int d1 , d2 ;`
	`} ;`
	`union Au {`
	`  struct Ss t1 ;`
	`} u = { 1 , 2 } ;`

(24)	**【漢字コード種別】** EUC コードを含むソースを Windows 上で使用するときは，環境変数 LANG78K を euc に設定するか，-ze オプションを指定してください。
(25)	**【セクションの開始アドレス指定】** #pragma section 指令で開始アドレスを指定したセクションのサイズは，常に偶数となります。

付録 C コマンド・オプション

この章では，プログラムのオプションを表形式でまとめました。

プログラム開発の際にお役立てください。

このオプション一覧は，索引としても使用することができます。

C.1 コンパイラ・オプション

表 C-1 コンパイラ・オプション

分類	記述形式	機能	他のオプションとの関係	省略時解釈
デバイス種別指定	-c *デバイス種別*	対象とするデバイス種別を指定します。	独立	省略することはできません。
オブジェクト・モジュール・ファイル作成指定	-o[*出力ファイル名*]	オブジェクト・モジュール・ファイルの出力を指定します。	-o と -no を同時に指定した場合は，あとで指定した方が有効です。	-o *入力ファイル名* .rel
	-no	オブジェクト・モジュール・ファイルを出力しない指定をします。		
メモリ配置指定	-r *処理種別*（複数指定可能）	メモリの割り付け方法を指定します。	-r と -nr，-rd と -nr，-rs と -nr を同時に指定した場合は，あとで指定した方が有効です。	-nr
	-rd[*n*][m]（*n* = 1, 2, 4）	外部変数／外部スタティック変数を自動的に saddr 領域に割り付けるように指定します。		
	-rs[*n*][m]（*n* = 1, 2, 4）	static auto 変数を自動的に saddr 領域に割り付けるように指定します。		
	-nr	-r，-rd，-rs オプションを無効にします。		
最適化指定	-q[*最適化種別*]（複数指定可能）	最適化フェーズを呼び出し効率の良いオブジェクトを生成するように指定します。	-q と -nq を同時に指定した場合は，あとで指定した方が有効です。	-qcjlvw
	-nq	-q オプションを無効にします。		
デバッグ情報出力指定	-g[*n*]（n = 1, 2）	ソース・レベルのデバッグの情報の出力を指定します。	-g と -ng を同時に指定した場合は，あとで指定した方が有効です。	-g2
	-ng	-g オプションを無効にします。		

分類	記述形式	機能	他のオプションとの関係	省略時解釈
プリプロセス・リスト・ファイル作成指定	-p[*出力ファイル名*]	プリプロセス・リスト・ファイルの出力を指定します。	-p が指定されていない場合 -k は無効です。	なし（ファイルを出力しない）
	-k[*処理種別*]（複数指定可能）	プリプロセス・リストに対する処理を指定します。		-kfln
プリプロセス指定	-d *マクロ名* [= *定義名*][, *マクロ名* [= *定義名*]] …（複数指定可能）	C ソース中で定義された文に対応する処理を指定します。	独立	C ソース・モジュール・ファイル中のマクロ定義のみを有効とします。
	-u *マクロ名* [, *マクロ名*] …（複数指定可能）	C ソース中の #undef 文と同様にマクロ定義を無効にします。	独立	-d で指定したマクロ定義を有効とします。
	-i *フォルダ*[, *フォルダ*] …（複数指定可能）	C ソース中の #include 文で指定されたインクルード・ファイルを指定されたフォルダから入力するよう指定します。	独立	1. ソース・ファイルのあるフォルダ 2. 環境変数 INC78K0R により指定されたフォルダ 3. C:¥Program Files¥NEC Electronics Tools¥CC78K0R¥V*x.xx*¥inc78k0r
アセンブラ・ソース・モジュール・ファイル作成指定	-a[*出力ファイル名*]	アセンブラ・ソース・モジュール・ファイルの出力を指示します。	-a と -sa が同時に指定された場合，-sa は無効です。	アセンブラ・ソース・モジュール・ファイルを出力しません。
	-sa[*出力ファイル名*]	アセンブラ・ソース・モジュール・ファイルにコメントとして C ソースを付加します。		
エラー・リスト・ファイル作成指定	-e[*出力ファイル名*]	エラー・リスト・ファイルを出力することを指定します。	独立	エラー・リスト・ファイルを出力しません。
	-se[*出力ファイル名*]	エラー・リスト・ファイルに C ソース・モジュール・ファイルを付加します。	独立	
クロスリファレンス・リスト・ファイル作成指定	-x[*出力ファイル名*]	クロスリファレンス・リスト・ファイルの出力を指定します。	独立	クロスリファレンス・リスト・ファイルを出力しません。

分類	記述形式	機能	他のオプションとの関係	省略時解釈
リスト形式指定	-lw[*文字数*]	各種リスト・ファイルの 1 行の文字数を指定します。	独立	-lw132（コンソール出力の場合は 80 文字とします）
	-ll[*行数*]	各種リスト・ファイルの 1 ページの行数を指定します。	独立	改ページしません。
	-lt[*文字数*]	ソース・モジュール中の HT（Horizontal Tabulation）コードを，各種リスト上でいくつかのブランク（空白）に置き換えて出力する（タビュレーション処理）ための基本となる文字数を指定します。	独立	-lt8
	-lf	各種リスト・ファイルの最後に改ページ・コードを付加することを指定します。	独立	改ページ・コードを付加しません。
	-li	C ソース・コメント付きアセンブラ・ソース・モジュール・ファイルに，インクルード・ファイルの C ソースも付加します。	独立	インクルード・ファイルの C ソースを付加しません。
ワーニング出力指定	-w[*レベル*]	ワーニング・メッセージをコンソールに出力するか否かを指定します。	独立	-w1
実行状態表示指定	-v	現在のコンパイルの実行状態をコンソールに出力するか否かを指定します。	-v と -nv を同時に指定した場合は，あとで指定した方が有効です。	-nv
	-nv	-v オプションを無効にします。		
パラメータ・ファイル指定	-f *ファイル名*	入力ファイル名，およびオプションを，指定したファイルより入力します。	独立	コマンド行上からのみオプション，入力ファイル名の入力が可能
テンポラリ・ファイル作成フォルダ指定	-t *フォルダ*	テンポラリ・ファイルを指定したドライブ，フォルダに作成します。	独立	環境変数 TMP により指定されたドライブ，フォルダ
ヘルプ指定	--/-?/-h	--/-?/-h オプションは，オプションの簡単な説明，デフォルト・オプションなどのヘルプ・メッセージをコンソールに表示します（コマンド・ラインのみ有効）	他のオプションをすべて無効にします。	表示しない

分類	記述形式	機能	他のオプションとの関係	省略時解釈
機能拡張指定	-z *種別*（複数指定可能）	拡張機能を有効にします。	-z と -nz を同時に指定した場合は，あとで指定した方が有効です。	-nz
	-nz	-z オプションを無効とします。		
デバイス・ファイルのサーチ・パス	-y *フォルダ*	デバイス・ファイルをサーチするパスを指定します。	独立	通常のサーチ・パスのみ
メモリ・モデル指定	-m *種別*	コンパイル時のメモリ・モデルの種類を指定します。	独立	-mm

総合索引

S

T

U

V

W

Y

Z

【あ行】

【か行】

【さ行】

【た行】

【は行】

【ら行】

〔メ　モ〕

【発　行】
NECエレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部1753
電話（代表）：044(435)5111

お問い合わせ先

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。
URL(アドレス)　http://www.necel.co.jp/

【営業関係，技術関係お問い合わせ先】

半導体ホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）
電　話　：044−435−9494
E-mail　：info@necel.com

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクスの販売特約店へお申し付けください。