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

CC78K0 Ver.3.70

Cコンパイラ

操作編

対象デバイス
78K0シリーズ

資料番号　U17201JJ1V0UM00（第1版）
発行年月　February 2005　CP(K)

〔メ　モ〕

目次要約

WindowsとWindowsNT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。

UNIXは，X/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標です。

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2005年2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，CC78K0（78K0シリーズ　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，CC78K0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，“CC78K0　C**コンパイラ　ユーザーズ・マニュアル　言語編**（U17200J）”（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】

CC78K0では，78K0シリーズ・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，RA78K0（78K0シリーズ　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　CC78K0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，および総合索引があります。

【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
CC78K0 Ver.3.70　Cコンパイラ	操作編	このマニュアル	U17201E
	言語編	U17200J	U17200E
RA78K0 Ver.3.80　アセンブラ・パッケージ	操作編	U17199J	U17199E
	言語編	U17198J	U17198E
	構造化アセンブリ言語編	U17197J	U17197E
SM+　システム・シミュレータ	操作編	U17246J	U17246E
	ユーザ・オープン・インタフェース編	U17247J	U17247E
SM78Kシリーズ Ver.2.52　システム・シミュレータ	操作編	U16768J	U16768E
PM plus Ver.5.20		U16934J	U16934E
ID78K0-NS Ver.2.52　統合ディバッガ	操作編	U16488J	U16488E
ID78K0-QB Ver.2.81　統合ディバッガ	操作編	U16996J	U16996E
78K0シリーズ	命令編	U12326J	U12326E

【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**シリーズ用 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

CC78K0 シリーズ　C コンパイラは，ANSI-C 注，または 78K0 シリーズの C 言語で記述された C ソース・プログラムを 78K0 シリーズ用の機械語に変換するプログラムです。

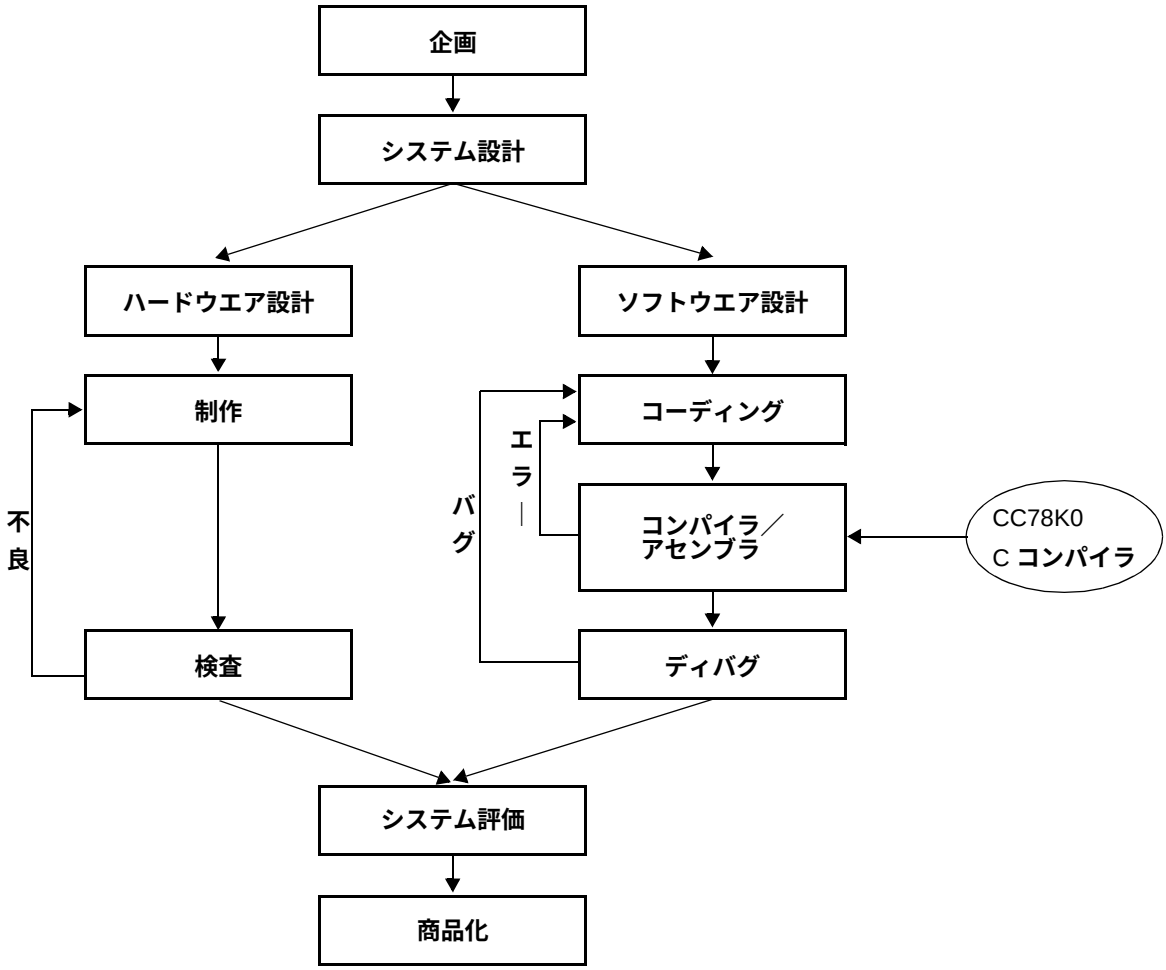
C コンパイラは，78K0 シリーズ用アセンブラ・パッケージ（RA78K0）に添付されている PM plus を利用することにより Windows98/Me/2000/XP，または WindowsNT™4.0 上で起動できるようになっています。PM plus を利用しない場合には，DOS プロンプト（Windows98/Me），またはコマンド・プロンプト（Windows2000/XP/NT4.0）上で起動することになります。

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

1.1　CC78K0 の役割

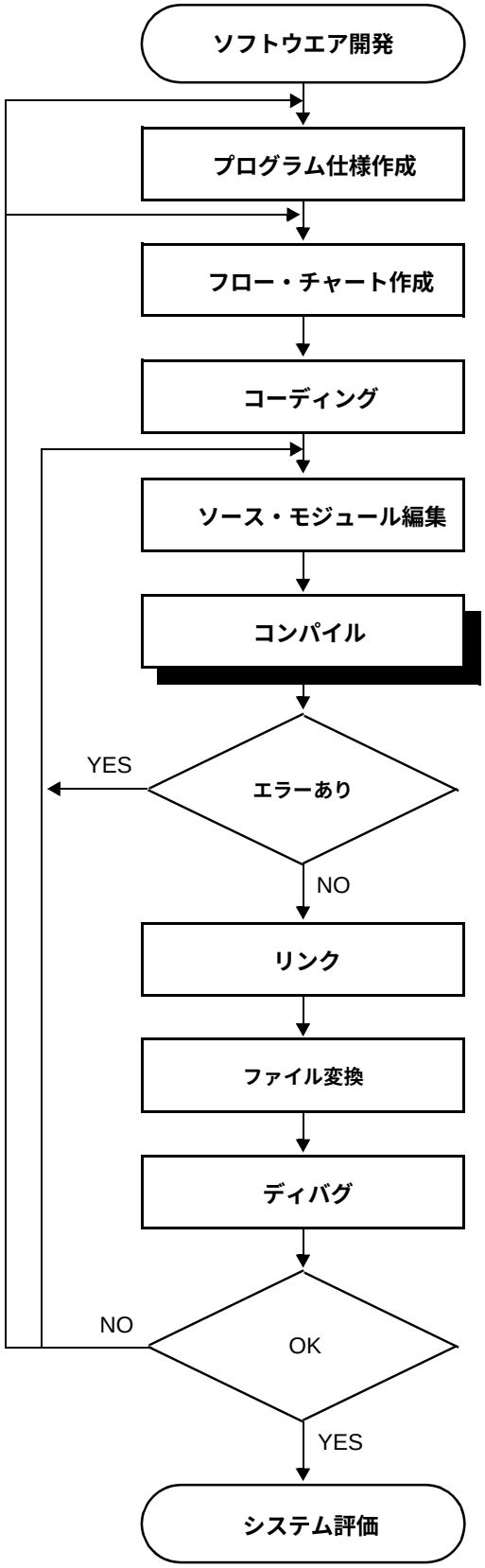
製品開発における，CC78K0 の位置付けを次に示します。

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



・・・ 78K 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。

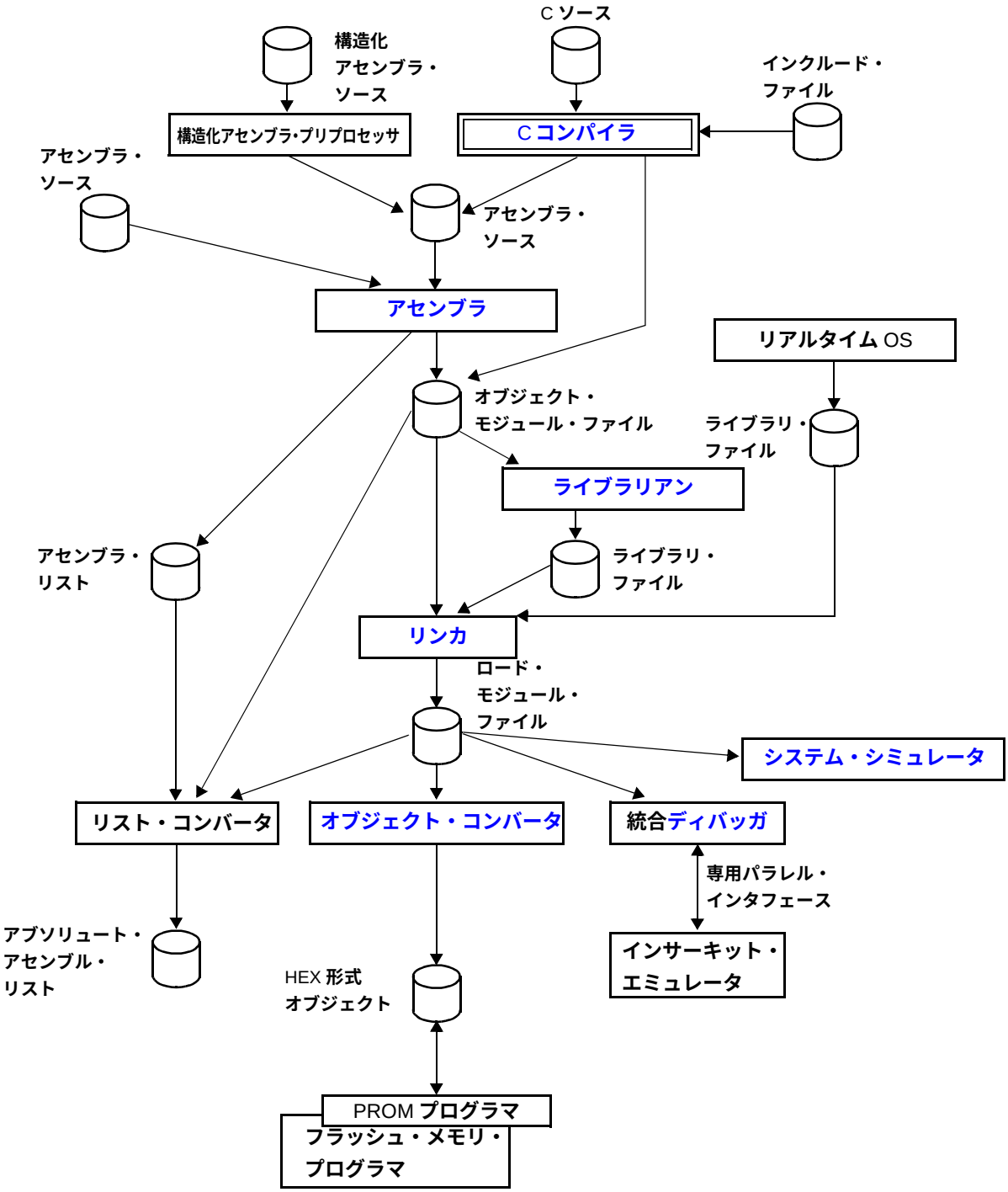
・・・ ファイルを HEX 形式に変換します。

・・・ ハードウエア・ディバッガ（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 CC78K0 による開発手順

CC78K0 における開発手順を次に示します。

図 1-3 CC78K0 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

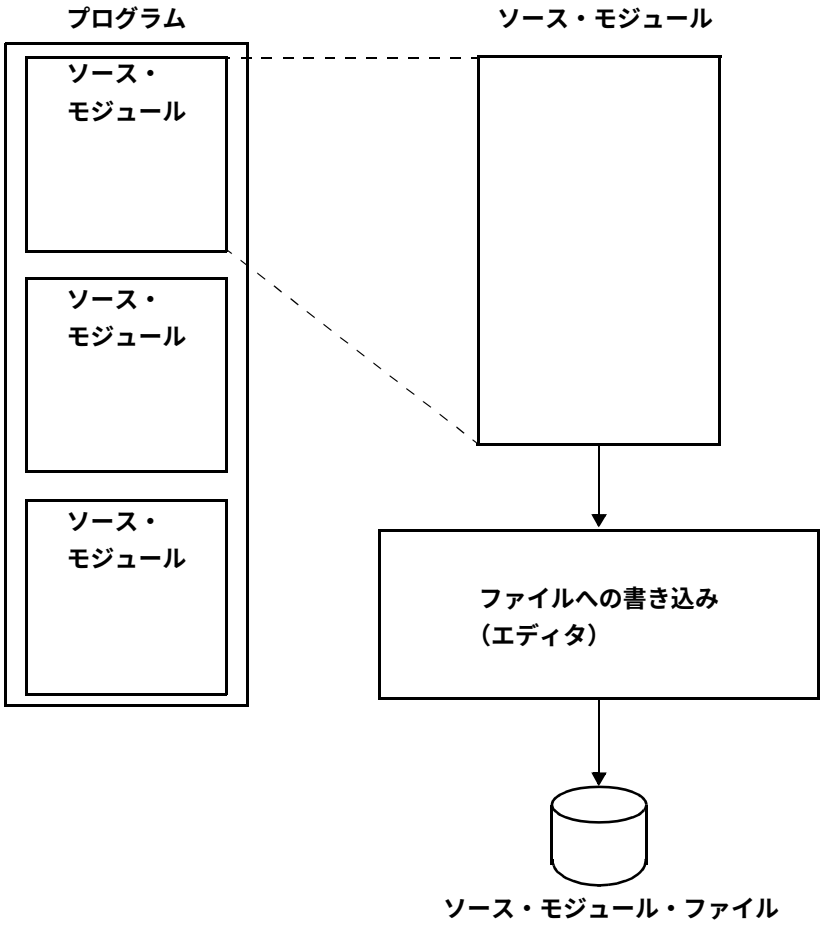
1 つのプログラムを機能的にいくつかに分割します。

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，CC78K0 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

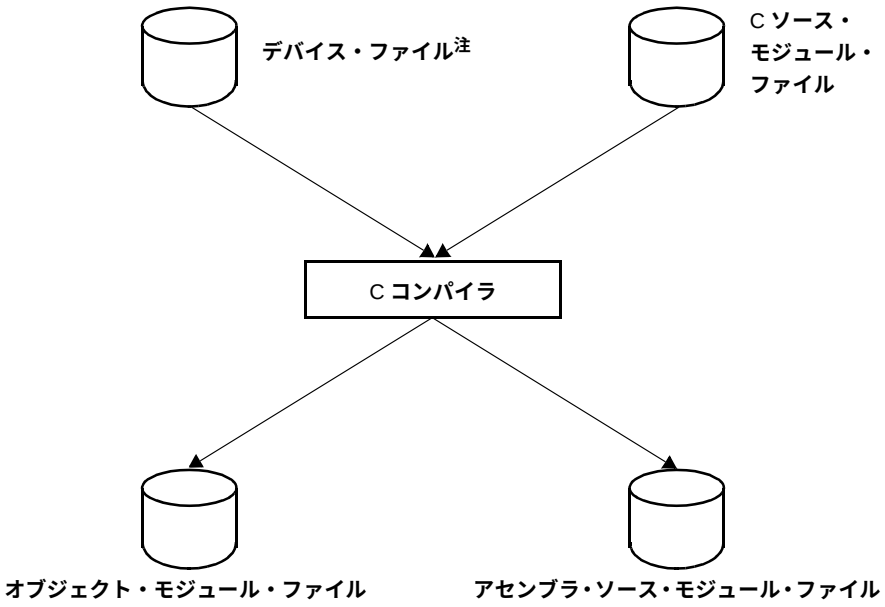


1.2.2 C コンパイラ

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 の「開発ツールダウンロード（ODS）」からジャンプできます。

http://www.necel.com/micro/ods/jpn/tool/DeviceFile/list.html

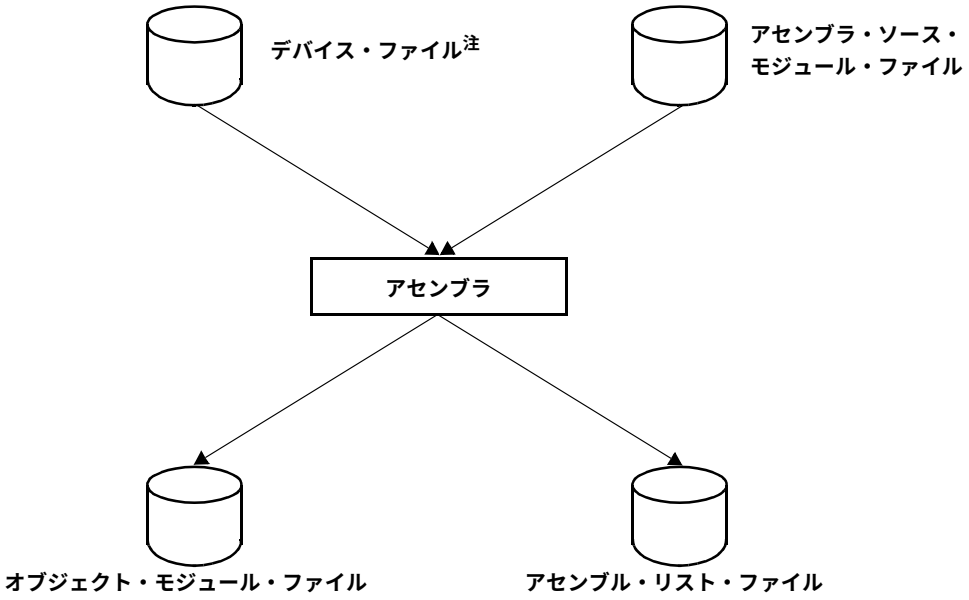
1.2.3 アセンブラ

アセンブルは，RA78K0 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翻訳するプログラムです。ソース・モジュー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 の「開発ツールダウンロード（ODS）」からジャンプできます。

http://www.necel.com/micro/ods/jpn/tool/DeviceFile/list.html

1.2.4 リンカ

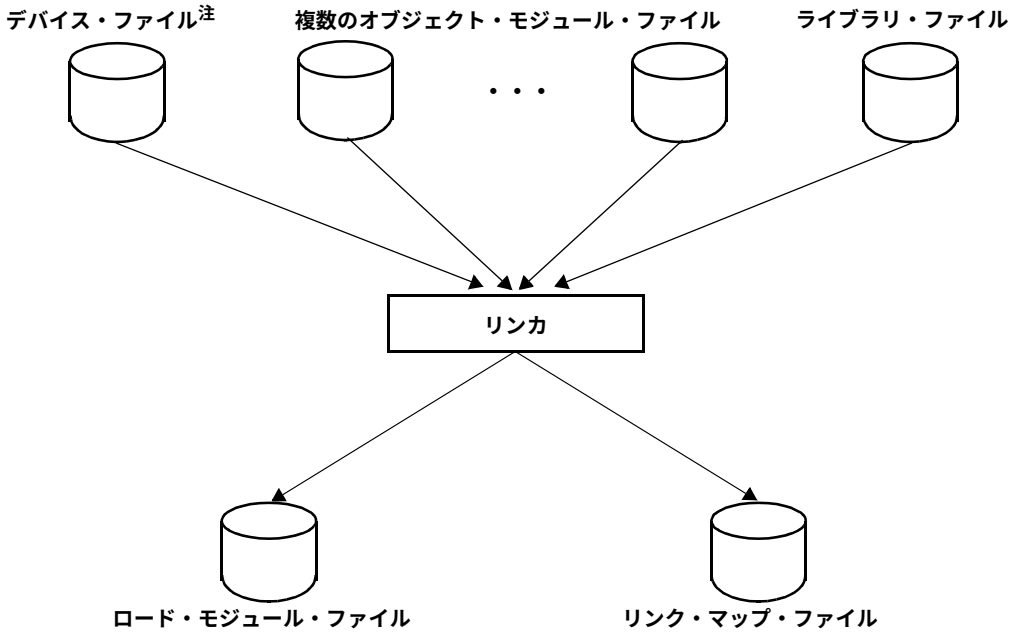
リンクは，RA78K0　アセンブラ・パッケージ（別売）に含まれているリンカを使用して行います。

リンカは，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 の「開発ツールダウンロード（ODS）」からジャンプできます。

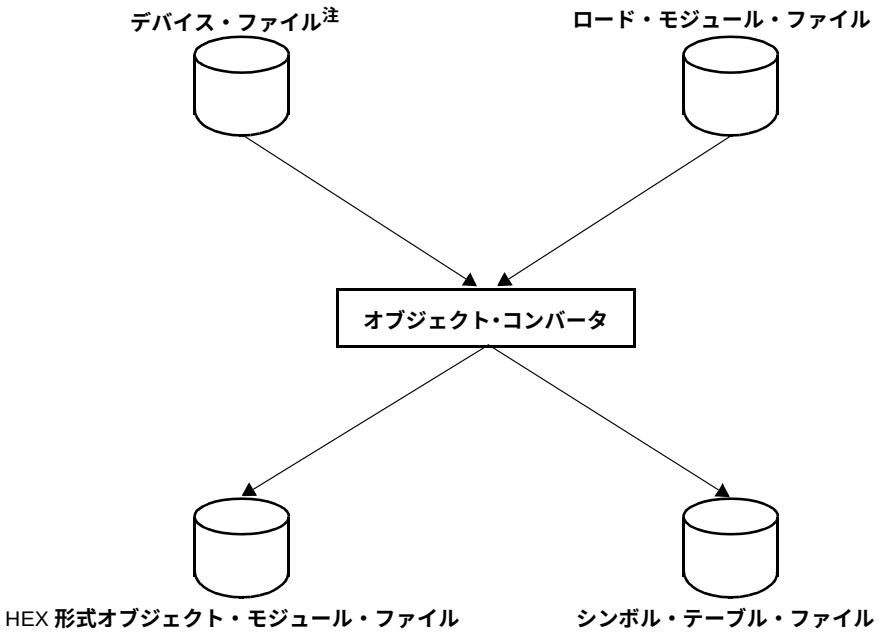
http://www.necel.com/micro/ods/jpn/tool/DeviceFile/list.html

1.2.5 オブジェクト・コンバータ

オブジェクト・コンバータは，RA78K0　アセンブラ・パッケージ（別売）に含まれているものを使用します。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，ファイル形式を変換します。そして，その結果をインテル標準 HEX 形式オブジェクト・モジュール・ファイルとして出力します。

また，シンボル情報をシンボル・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 の「開発ツールダウンロード（ODS）」からジャンプできます。

http://www.necel.com/micro/ods/jpn/tool/DeviceFile/list.html

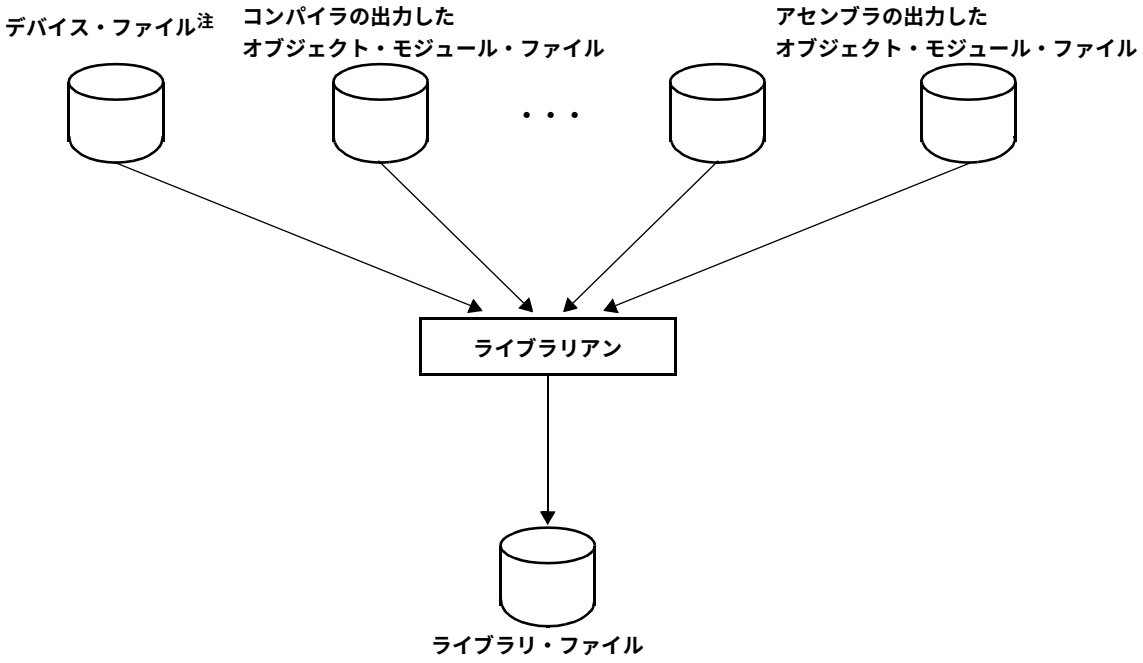
1.2.6 ライブラリアン

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・ファイルの作成と更新にライブラリアンを使用します。ライブラリアンは，RA78K0　アセンブラ・パッケージ（別売）に含まれているものを使用します。

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 の「開発ツールダウンロード（ODS）」からジャンプできます。

http://www.necel.com/micro/ods/jpn/tool/DeviceFile/list.html

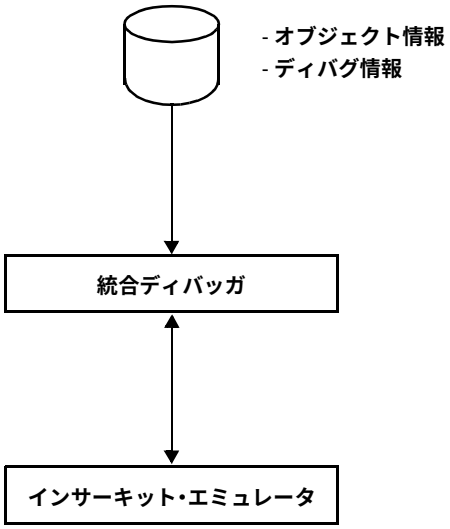
1.2.7 ディバッガ

リンカが出力したロード・モジュール・ファイルを ID78K0-NS / ID78K0-QB（78K0 シリーズ用統合ディバッガ）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，グラフィカルなユーザ・インタフェースを用いたソース・ディバグが可能となります。

ディバグのために，対象のソース・プログラムをコンパイルする際にディバグ情報出力指定オプション -G を指定しておきます（-G はディフォールト・オプションです）。その指定で，ディバ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）。

ディバッガ，IE は，別パッケージ（別売）になります。

図 1-10 ディバッガの機能



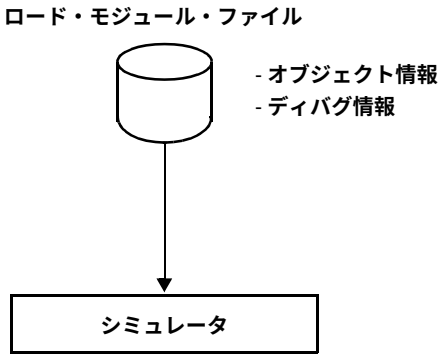
1.2.8 システム・シミュレータ

リンカが出力したロード・モジュール・ファイルを SM78K0（78K0 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，グラフィカルなユーザ・インタフェースを用いたソース・ディバグが可能となります。

SM78K0 は，ID78K0-NS / ID78K0-QB と同じ操作イメージで，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。SM78K0 では，機械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できます。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ますので，ターゲット・システムの動作を含めたプログラムのディバグを，ハードウエア開発から独立して早期に行えます。

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

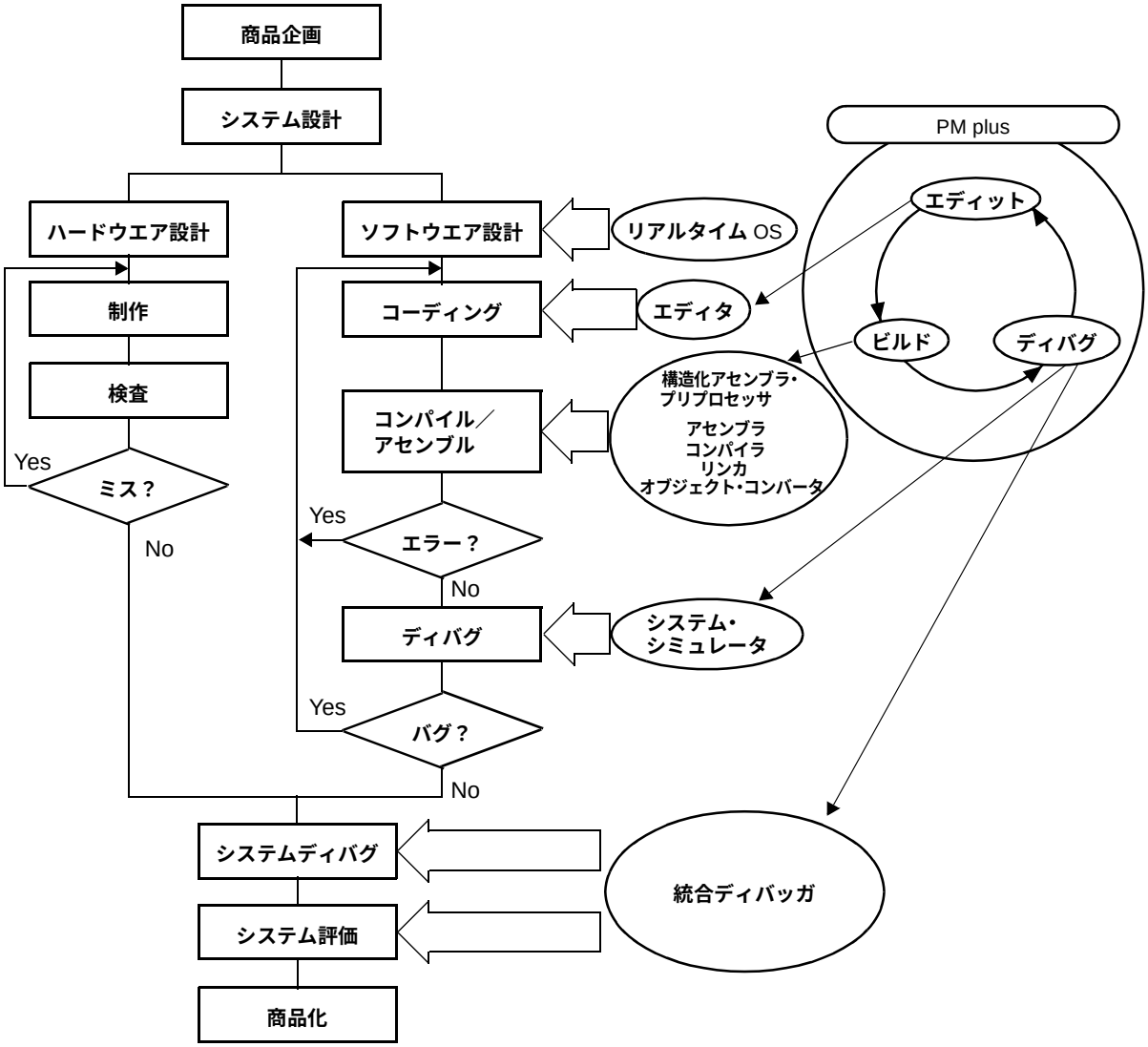


1.2.9 PM plus

PM plus は，CC78K0 に添付されている DLL ファイルを使用し，CC78K0 を Windows98/Me/2000/XP，または WindowsNT4.0 上で起動できるようにするソフトウエアです。PM plus の起動画面から，ソースのエディット，MAKEFILE の自動生成，およびコンパイルからリンクまでを行えます。これにより，GUI イメージでエディットからディバグまでが可能となります。

なお，PM plus は，RA78K0　アセンブラ・パッケージ（別売）に添付されます。インストールと設定は，RA78K0　アセンブラ・パッケージのインストーラを使用して行います。CC78K0 を PM plus から起動する場合は，コンパイラをインストールする前に，RA78K0　アセンブラ・パッケージをインストールしてください。

図 1-12 PM plus の機能



備考　ビルドでは，メイク・ファイルを解析実行し，実行可能なファイルを生成します。メイク・ファイルに記述された依存関係を基に不要なアセンブル，コンパイル，リンクなどは省略され，効率良く実行ファイルを生成できます。

第 2 章　製品概要とインストール方法

この章では，CC78K0 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・マシンと供給媒体

この C コンパイラは，表 2-1 に示す開発環境に対応しています。

表 2-1 C コンパイラの供給媒体と記録形式

ホスト・マシン	OS	供給媒体
IBM PC/AT™ 互換機	日本語 Windows（98/Me/2000/XP/NT4.0）注 英語 Windows（98/Me/2000/XP/NT4.0）注	CD-ROM

注　C コンパイラを Windows 上で使用するためには，PM plus が必要です。PM plus を使用しない場合は，DOS プロンプト（Windows98/Me），またはコマンド・プロンプト（Windows2000/XP/NT4.0）で C コンパイラを起動できます。

2.2 インストール

以下に，CC78K0 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

CC78K0 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，フォルダ CC78K0\DISK1 に格納されている SETUP.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 Explorer などを用いて，CC78K0 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 ディレクトリ構成」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 の「開発ツールダウンロード（ODS）」からジャンプできます。

http://www.necel.com/micro/ods/jpn/tool/DeviceFile/list.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，CC78K0 とともにインストールされます。

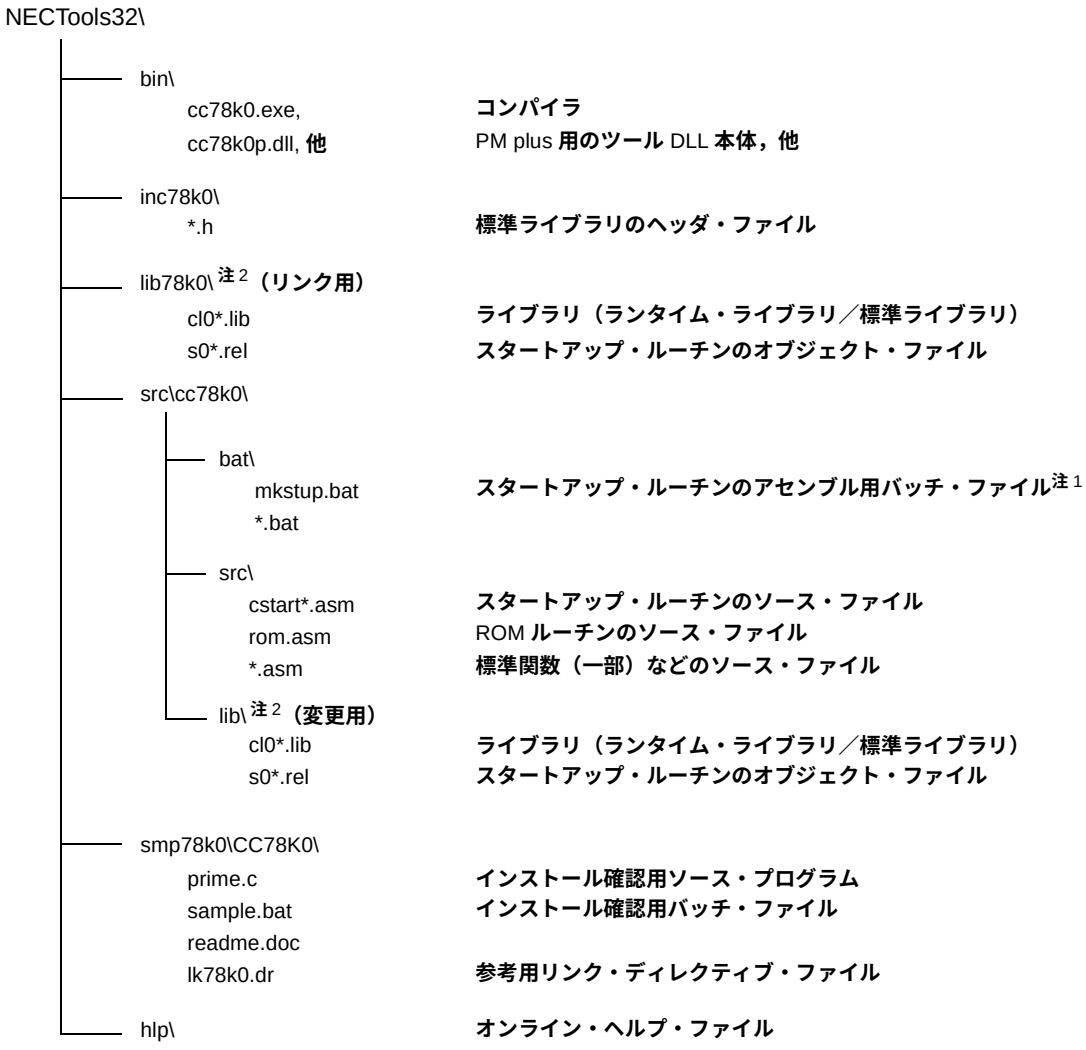
2.4 ディレクトリ構成

インストール時に表示される標準ディレクトリは，Windows システムの “ NECTools32 ” です。インストール・ディレクトリ下の構成は，次のようになります。ただし，ドライブやインストール・ディレクトリはインストール時に変更してもかまいません。

PM plus による MAKE を行う場合は，各ツール（CC78K0, RA78K0 など）を同じドライブ，およびディレクトリ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・プログラムのディフォールトの指示に従って，ディフォールトのプログラム名 “ NECTools32 ” で標準ディレクトリにインストールしたこととして説明しています。

図 2-1 ディレクトリ構成



注 1 このバッチ・ファイルは，PM plus 上では使用できません。バッチ・ファイルを使用する場合は，DOS プロンプト（Windows98/Me），またはコマンド・プロンプト（Windows2000/XP/NT4.0）で実行してください。

注 2 スタートアップ・ルーチンを修正する場合は，src\cc78k0\lib ディレクトリ以下のソース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 src\cc78k0\lib に入るので，lib78k0 ディレクトリにコ

ピーしてリンクしてください。

2.5 アンインストール手順

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) ［コントロールパネル］ウインドウのオープン

［スタート］ボタンの［設定 (S)］メニュー→［コントロールパネル (C)］を選択し，［コントロールパネル］ウインドウをオープンします。

(3) ［アプリケーションの追加と削除］ウインドウのオープン

［コントロールパネル］ウインドウの［アプリケーションの追加と削除］アイコンをダブル・クリックし，［アプリケーションの追加と削除］ウインドウをオープンします。

注意 Windows XP の場合，［アプリケーションの追加と削除］は［プログラムの追加と削除］となります。

(4) CC78K0 の削除

［アプリケーションの追加と削除］ウインドウの［セットアップと削除］タブに表示されているインストール済みソフトウエア一覧の中から “ NEC CC78K0 78K/0 C コンパイラ Vx.xx ” を選択した後，［追加と削除 (R) ］ボタンを押下します。

なお，［ファイル削除の確認］ウインドウがオープンした際には，［はい (Y)］ボタンを押下します。

(5) ファイル群の確認

Windows の標準アプリケーション Explorer 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ます。なお，各フォルダについての詳細は，「2.4 ディレクトリ構成」を参照してください。

2.6 環境設定

2.6.1 ホスト・マシン

CC78K0 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。

32 ビット化は，DOS Extender を使用する形で実現されておりますので，次の OS で動作するように設計されています。

Windows98/Me/2000/XP/NT4.0 Windows98/Me の DOS プロンプト Windows2000/XP/NT4.0 のコマンド・プロンプト

2.6.2 環境変数

DOS プロンプト（Windows98/Me），またはコマンド・プロンプト（Windows2000/XP/NT4.0）での作業時には，次の環境変数を設定してください。

表 2-2 環境変数

環境変数	説明
PATH	コンパイラを置いたディレクトリを指定します。
TMP	一時ファイルを作成するディレクトリを指定します。
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（ディフォールト） euc　EUC none　2 バイト・コードなし
INC78K0	コンパイラの標準ヘッダ・ファイルを置いたディレクトリを指定します。
LIB78K0	コンパイラのライブラリを置いたディレクトリを指定します。

【指定例】

```
PATH = %PATH% ; C:\NECTools32\bin
set      TMP = C:\
set      LANG78K = sjis
```

2.6.3 ファイル構成

各ディレクトリの内容を表 2-3 に示します。

ディレクトリ構造，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 2-3 ファイル構成（* = 英数字）

ディレクトリ名	ファイル名	概要
bin\	cc78k0.exe	コンパイラ本体
	cc78k0.msg	メッセージ・ファイル
	*.hlp	ヘルプ・ファイル
	*.dll	DLL ファイル
inc78k0\	*.h 注 1	標準ライブラリのヘッダ・ファイル
src\cc78k0\bat\ 注 2	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
	*.bat 注 3	標準関数（一部）などの更新用バッチ・ファイル
src\cc78k0\src	cstart*.asm 注 4 rom.asm *.asm 注 5	スタートアップ・ルーチンのソース・ファイル ROM 化ルーチンのソース・ファイル 標準関数（一部）などのソース・ファイル
hlp	*.chm	オンライン・ヘルプ・ファイル

注 1 「CC78K0 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 このディレクトリにあるバッチ・ファイルは，PM plus 上では使用できません。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，ご使用ください。

注 3 表 8-1 を参照してください。

注 4 * = 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 5 表 8-2 を参照してください。

2.6.4 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。

表 2-4 に，ディレクトリの内容を示します。

表 2-4 ライブラリ・ファイル

ディレクトリ名	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	
lib78k0\	cl00.lib cl00r.lib cl00sm.lib cl00f.lib	cl00.lib cl00r.lib cl00sm.lib cl00f.lib	cl00e.lib cl00re.lib cl00sme.lib cl00fe.lib	ライブラリ（ランタイム・ライブラリ，標準ライブラリ）注 1 （乗除算命令なしの品種の場合）
	cl0.lib cl0r.lib cl0sm.lib cl0f.lib cl0x.lib 注 3 cl0xr.lib 注 3 cl0xsm.lib 注 3	cl0.lib cl0r.lib cl0sm.lib cl0f.lib cl0x.lib 注 3 cl0xr.lib 注 3 cl0xsm.lib 注 3	cl0e.lib cl0re.lib cl0sme.lib cl0fe.lib cl0xe.lib 注 3 cl0xre.lib 注 3 cl0xsme.lib 注 3	ライブラリ（ランタイム・ライブラリ，標準ライブラリ）注 1 （乗除算命令ありの品種の場合）
	s0.rel s0l.rel s0sm.rel s0sml.rel	s0b.rel s0lb.rel s0smb.rel s0smlb.rel	s0e.rel s0le.rel s0sme.rel s0smle.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　ライブラリの命名規則は，次のようになっています。

lib78k0\cl0< mul/div >< float >< pascal >< model >< flash >.lib

< mul/div >

なし　乗除算器未使用

x　　乗除算器使用

< 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

< pascal >

なし　通常関数インタフェース使用時

r　　パスカル関数インタフェース使用時（コンパイル・オプション -ZR 指定時）

< model >

なし　ノーマル・モデル

sm　スタティック・モデル

< 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。**

lib78k0\s0< model >< lib >< flash >.rel

< model >

なし　ノーマル・モデル

sm　**スタティック・モデル**

< 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

b　　**ブート領域用**

e　　**フラッシュ領域用**

注３　CC78K0 **のライブラリでは，次のようなデバイスの乗除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，「CC78K0 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
剰余データ・レジスタ 0	SDR0	FF60H	16 bit
乗除算データ・レジスタ A0	MDA0H , MDA0L	FF64H , FF62H	16 bit × 2
乗除算データ・レジスタ B0	MDB0	FF66H	16 bit
乗除算器コントロール・レジスタ 0	DMUC0	FF68H	8 bit

＜乗算時のレジスタ構成＞

＜乗数 A **＞　　＜乗数** B **＞　　＜積＞**

MDA0**（ビット** 15-0**）**×　MDB0**（ビット** 15-0**）**= MDA0**（ビット** 31-0**）**

＜除算時のレジスタ構成＞

＜被除数＞ ＜除数＞ ＜商＞ ＜剰余＞

MDA0（ビット 31-0）÷ MDB0（ビット 15-0）= MDA0（ビット 31-0）… SDR0（ビット 15-0）

＜乗除算器コントロール・レジスタ 0 ＞

7	6	5	4	3	2	1	0
DMUE	0	0	0	0	0	0	DMUSEL0

DMUE ： 演算動作の停止（0）／開始（1）

DMUSEL0 ： 除算モード（0）／乗算モード（1）

備考 ビット番号を四角で囲んでいるものは，そのビット名称が RA78K0 では予約語に，CC78K0 では #pragma sfr 指令で，sfr 変数として定義されているものです。

第 3 章　コンパイルからリンクまでの手順

この章では，CC78K0 と RA78K0　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 plus 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 plus について

ここでは，RA78K0　アセンブラ・パッケージに含まれる PM plus で CC78K0 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 plus から CC78K0 を起動する場合，CC78K0 に含まれる cc78k0p.dll が参照されます。

3.1.1　cc78k0p.dll（ツール DLL）の位置づけ

cc78k0p.dll ファイルなどのツール DLL ファイルは，PM plus 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 plus に準じます。

3.1.3 CC78K0 用オプション設定メニュー

(1) **オプション・メニュー項目**

CC78K0 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 plus **の［ツール** (T)**］メニュー中に次の項目が追加されます。**

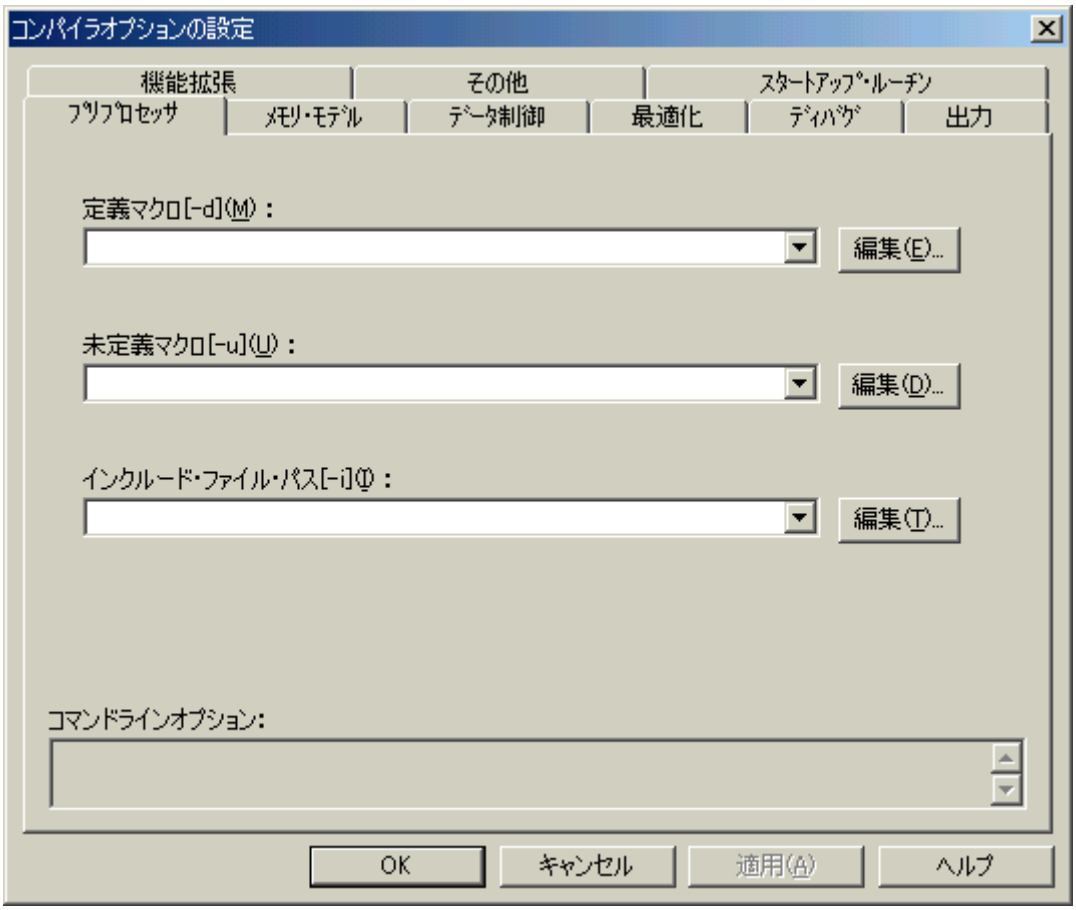
［コンパイラオプションの設定 (C)**］**

(2) **〈コンパイラオプションの設定〉ダイアログ**

PM plus **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ばれます。**

〈コンパイラオプションの設定〉ダイアログは次のとおりです。

図 3-1 **〈コンパイラオプションの設定〉ダイアログ**



(3) **参照ダイアログ**

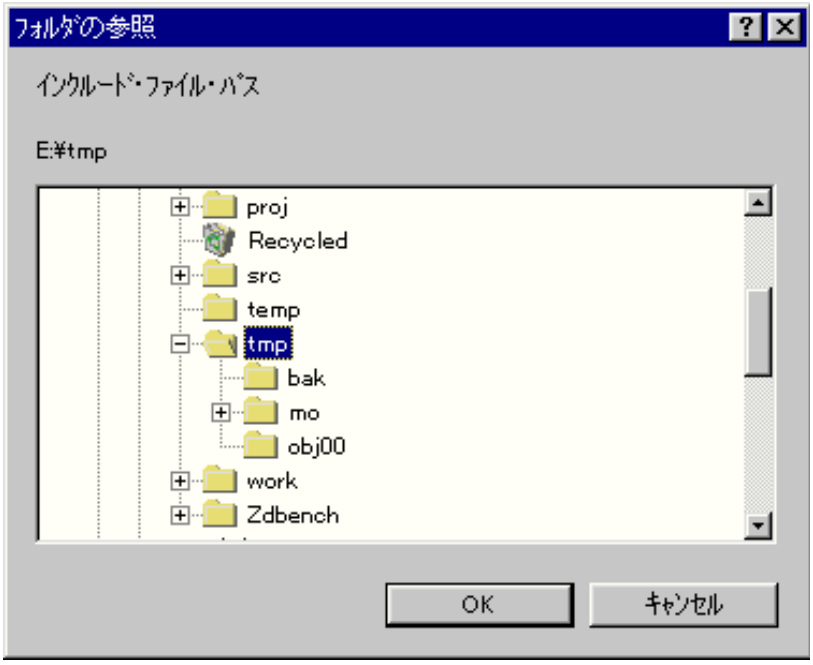
〈コンパイラオプションの設定〉ダイアログで，以下のパス設定に対して［参照］ボタンを押すと，次のダイアログが表示されます。

なお，このダイアログではフォルダのみ指定できます。

- **オブジェクト・モジュール・ファイルの出力パス**
- **アセンブラ・モジュール・ファイルの出力パス**
- **エラー・リスト・ファイルの出力パス**

- クロス・レファレンス・リスト・ファイルの出力パス
- プリプロセス・リスト・ファイルの出力パス
- テンポラリ・ファイル・パス

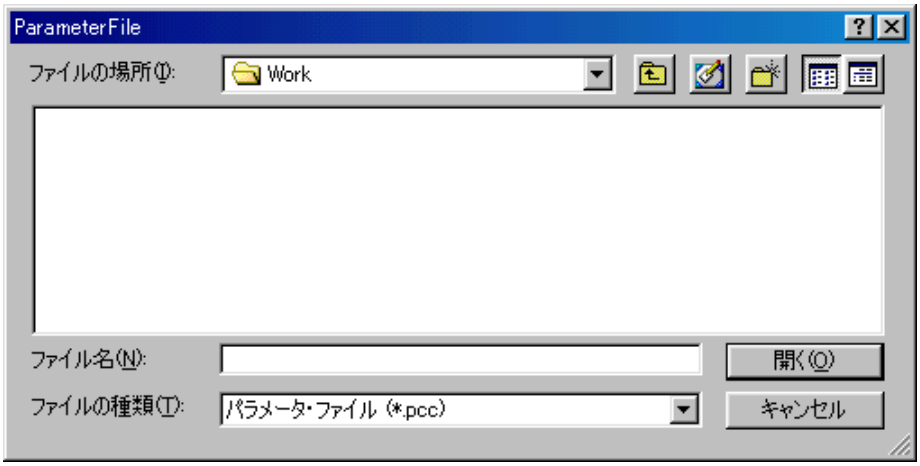
図 3-2 〈フォルダの参照〉ダイアログ



パラメータ・ファイル指定で［参照］ボタンを押すと，次のダイアログが表示されます。
また，このダイアログのディフォールトは次の通りです。

カレント・ディレクトリ：プロジェクト・ファイルのディレクトリ
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



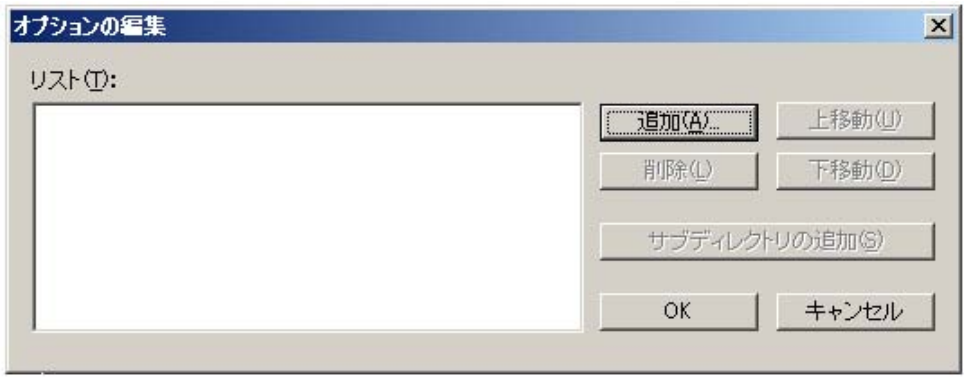
(4) **〈オプションの編集〉ダイアログ**

〈コンパイラオプションの設定〉ダイアログで，［編集］ボタンを押すと，表示されます。

〈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。

〈オプションの編集〉ダイアログについて，次に説明します。

図 3-4 **〈オプションの編集〉ダイアログ**



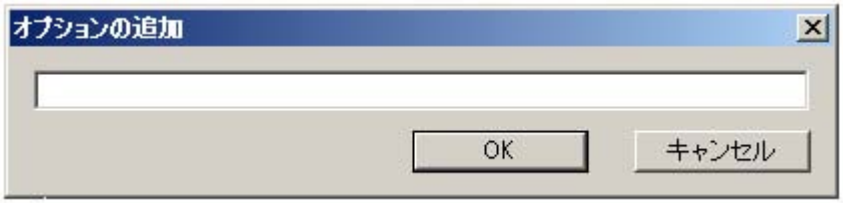
- **［追加** (A)**］ボタン**

  リストの項目を追加します。

  ファイルやディレクトリを指定する項目の場合は，それぞれの〈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**

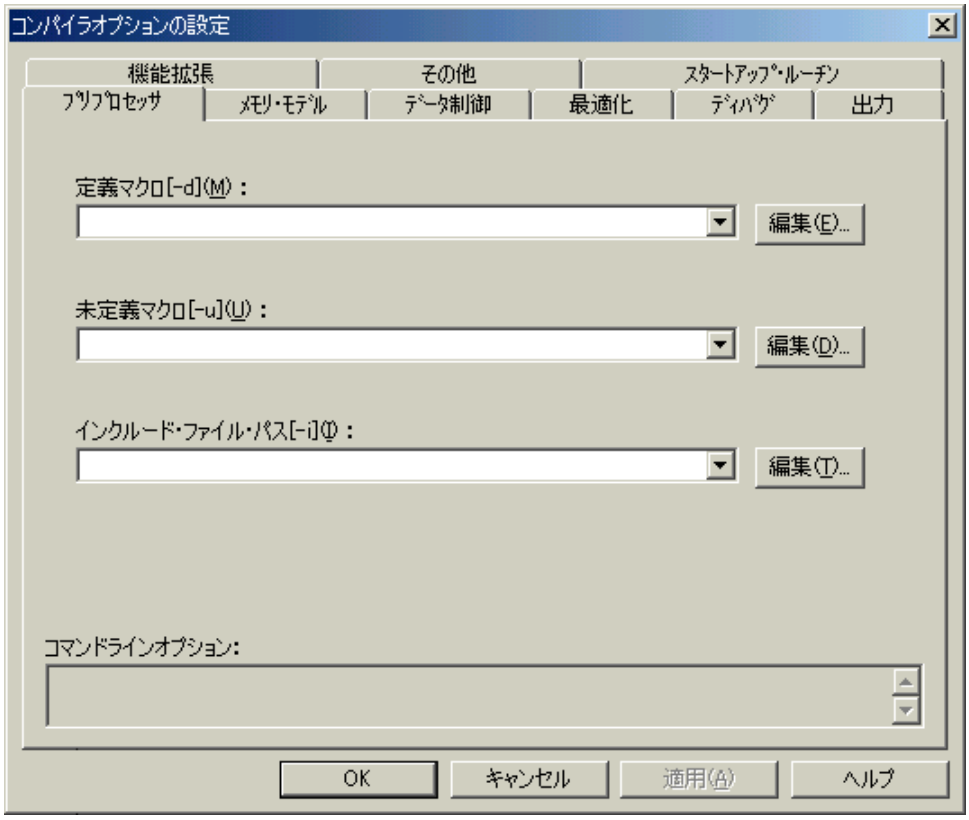
  次の場合は，選択中のリストの項目にサブディレクトリを追加できます。

  インクルード・ファイル・パス [-i](I)

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（ディフォールト）
  [メモリ・モデル] タブ
  [データ制御] タブ
  [最適化] タブ
  [ディバグ] 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**

  現在設定されているオプション文字列を表示します。

  〈その他〉設定ダイアログの“ その他のオプション (I) ”で入力し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，入力できません。このコンパイラのディフォールト・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できます。

- **［**OK**］ボタン**

  このダイアログ・ボックス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。Project Window **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・ボックスのどこにあっても，［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。

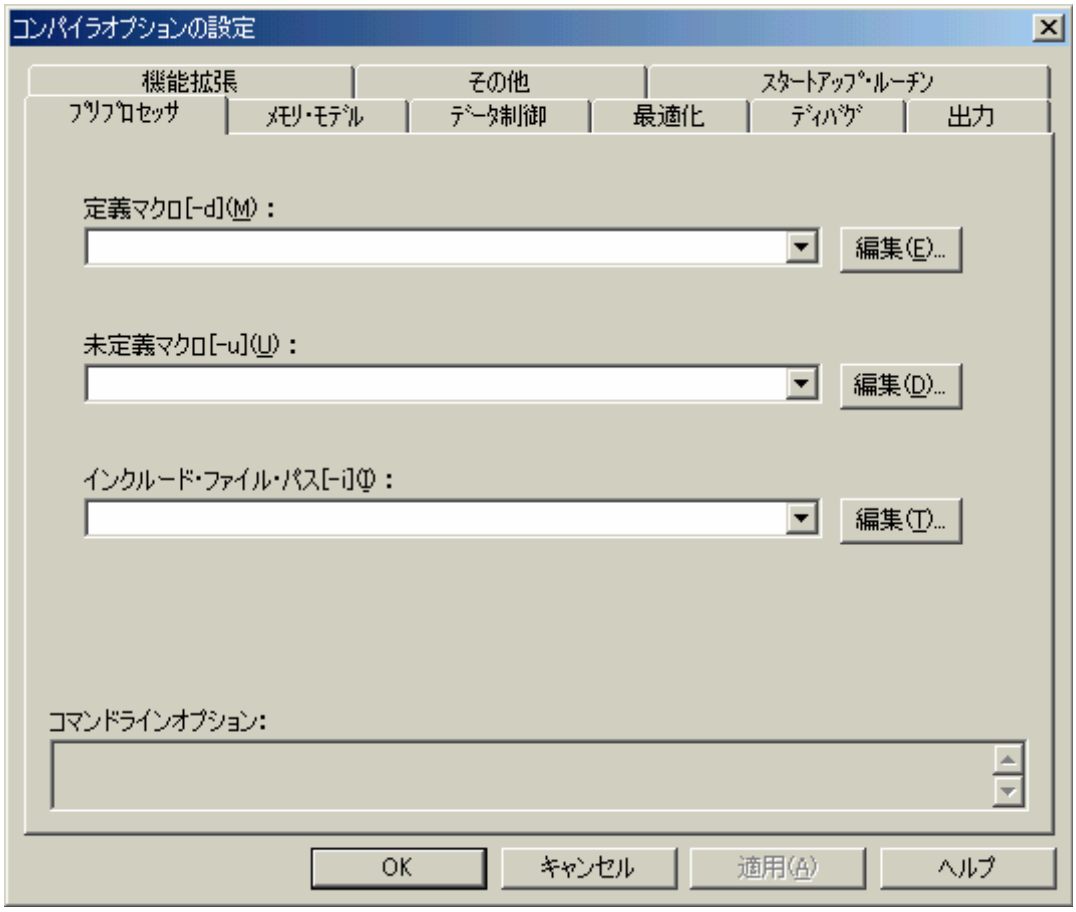
  このダイアログ・ボックスで編集した設定を決定し，〈コンパイラオプションの設定〉ダイアログを開いたままにします。

- **［ヘルプ］ボタン**

  このダイアログ・ボックスに関するヘルプ・メッセージを表示します。

(1) **［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，一度に複数のマクロ定義を行えます。

  ［編集 (E)**］ボタンで指定することもできます。**

- **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，一度に複数のマクロ定義を無効にできます。

  ［編集 (D)**］ボタンで指定することもできます。**

- **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・ファイルがあるディレクトリを，コンボ・ボックスに入力します。

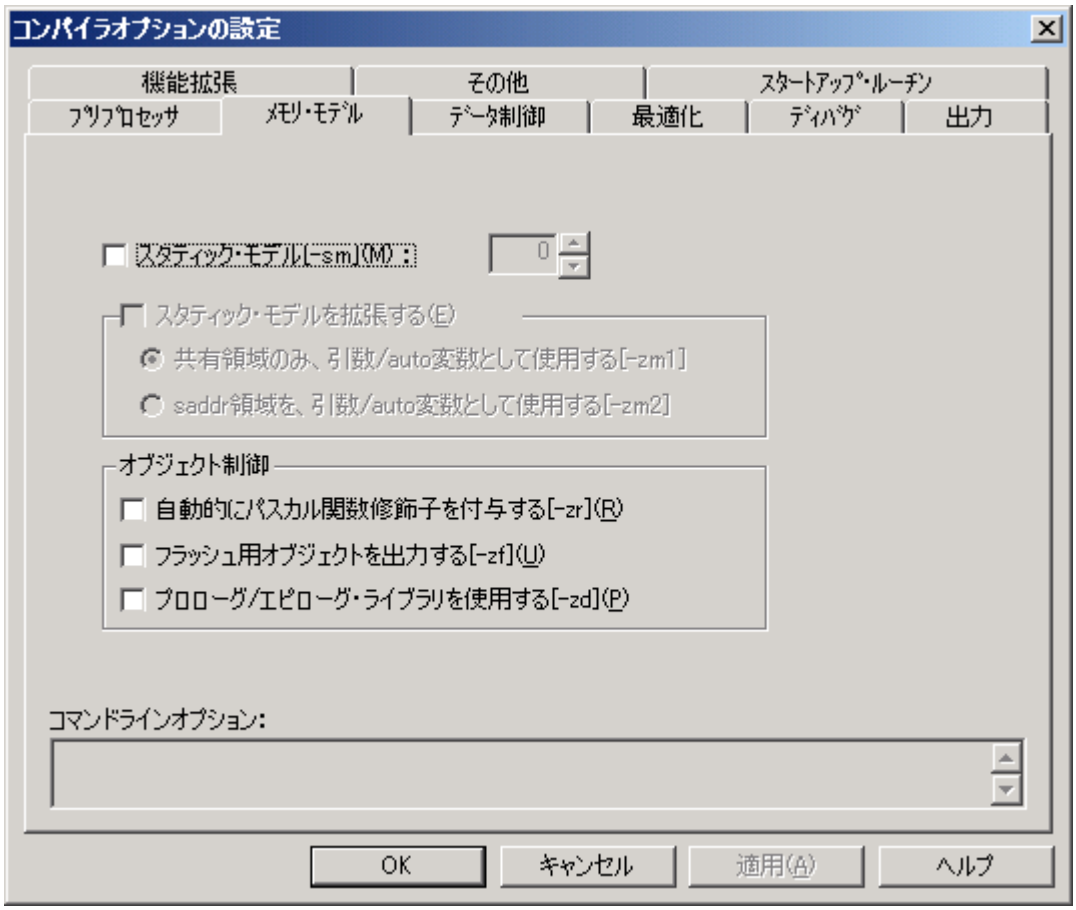
  ”, ” で区切ることにより，一度に複数のディレクトリを指定できます。

  ［編集 (T)**］ボタンで指定することもできます。**

  存在しないパスは，指定できません。

(2) ［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）



- **スタティック・モデル** [-sm](M)

  スタティック・モデルを使用する場合は，チェック・ボックスで選択し，共有領域のバイト数を指定します。

- **スタティック・モデルを拡張する** (E)

  -SM **オプションが指定されていて，スタティック・モデルを拡張したい場合に，チェック・ボックスをチェックします。**

  引数／ auto **変数として使用する領域を，ラジオ・ボタンをチェックすることにより選択します。**

  チェック・ボックスが未チェックでも，選択しているラジオ・ボタンの情報は記憶されます。

- **オブジェクト制御**

  自動的にパスカル関数修飾子を付与する [-zr](R)

  -ZR **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。**

  フラッシュ用オブジェクトを出力する [-zf](U)

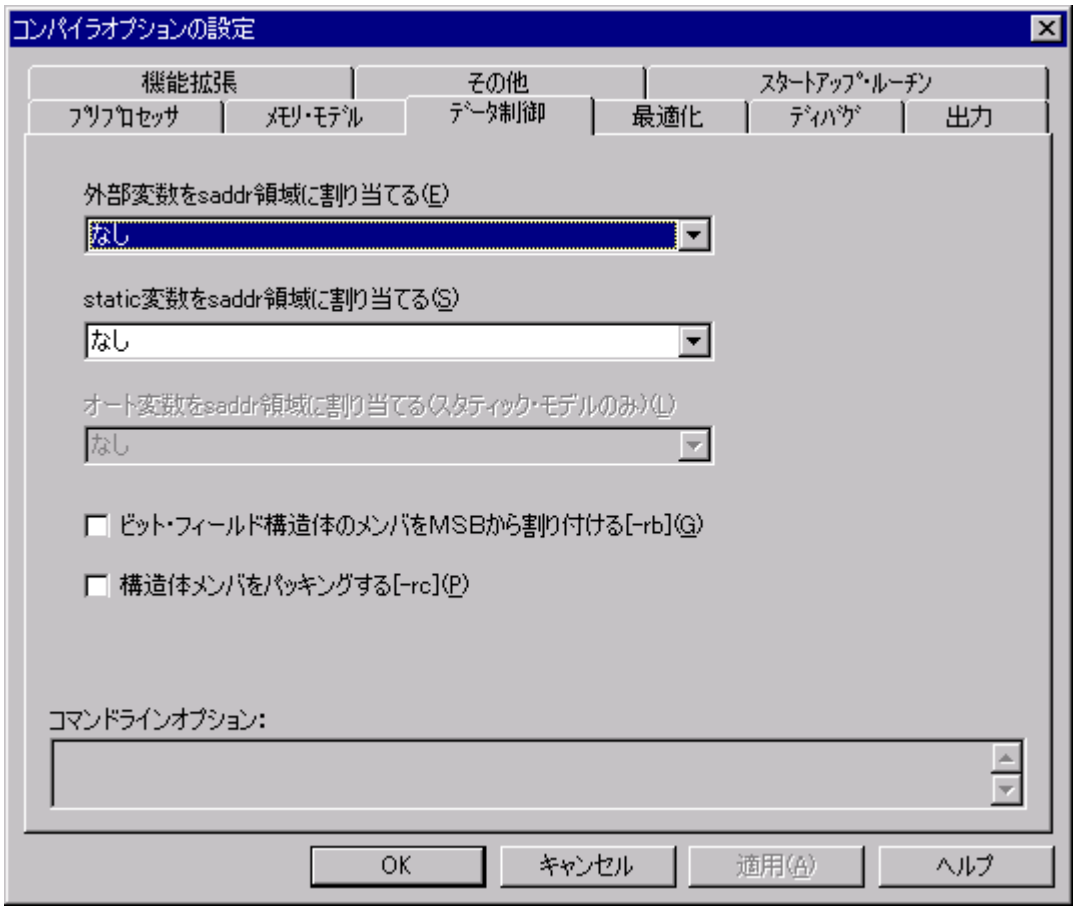
  -ZF **オプションを有効にする場合にチェック・ボックスをチェックします。**

  プロローグ／エピローグ・ライブラリを使用する [-zd](D)

  -ZD **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) ［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。**

- 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。**

- **オート変数を** saddr **領域に割り当てる（スタティック・モデルのみ）**(L)

  saddr **領域に配置させるオートマティック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。**

- **構造体メンバをパッキングする** [-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。**

(4) **［最適化］タブ**

(a) **“ グループ (G) ” で “ 推奨統合オプション (I) ” 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 (I) ” 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，ディフォールト，サイズ優先の 3 パターンがあり，それぞれの意味は次の通りです。

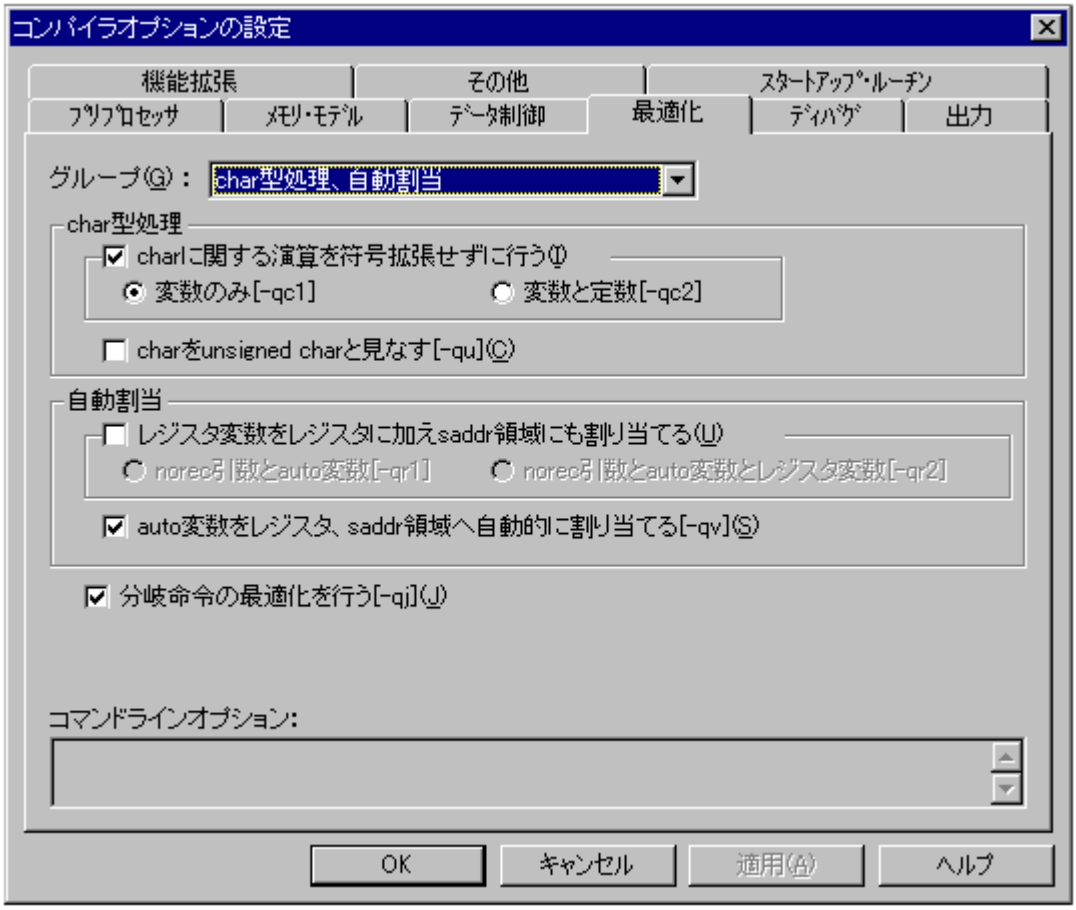
  スピード優先 [-qx1] ： -QX1 **オプションで，特に実行スピード効率を重視する場合に選択します。**

  ディフォールト [-qx2] ： -QX2 **オプションで，実行スピード，サイズ効率の両方とも重視する場合に選択します。**

  サイズ優先 [-qx3] ： -QX3 **オプションで，特にオブジェクト・コード・サイズ効率を重視する場合に選択します。**

(b) “ 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** (I)

  -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。符号拡張しない** char **型演算数の種類を，ラジオ・ボタンをチェックして選択します。**

  char **を** unsigned char **とみなす** [-qu](C)

  -QU **オプションを有効にしたい場合にチェック・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。**

(c) “ 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

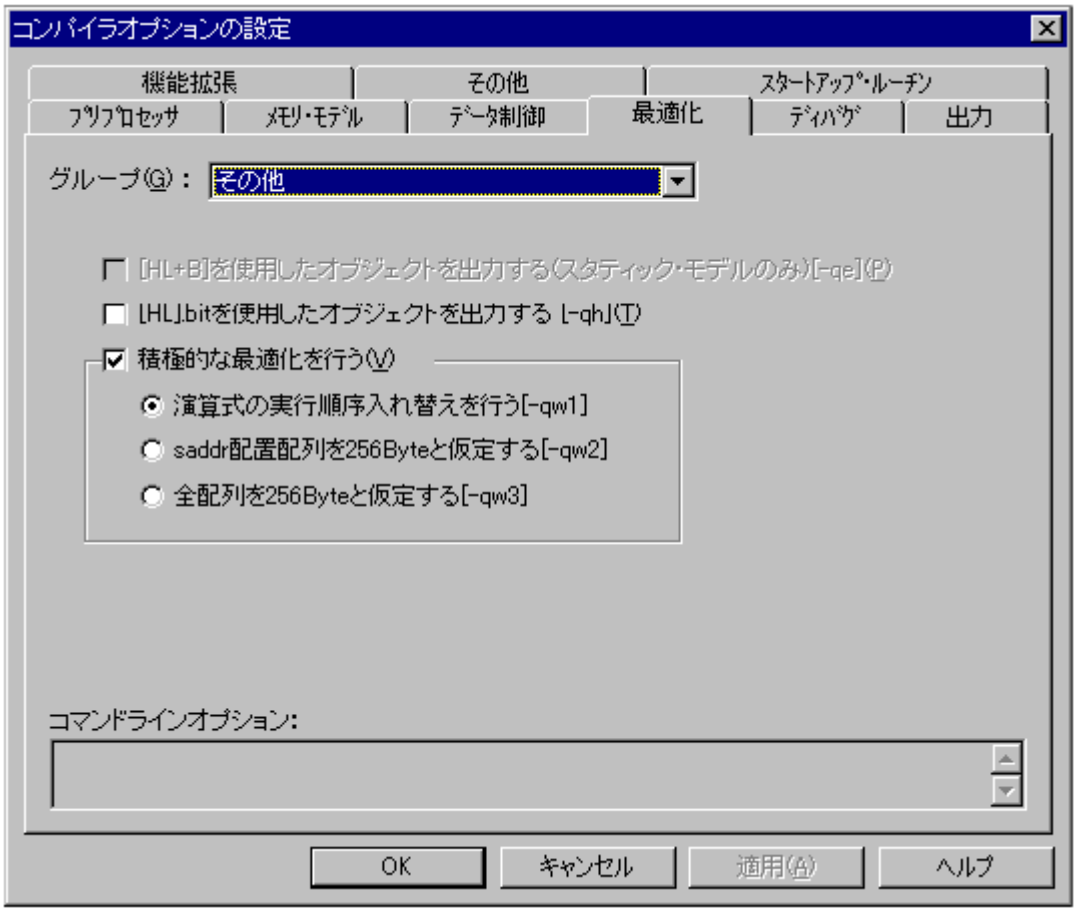
図 3-12 〈コンパイラオプションの設定〉ダイアログ（“ ライブラリ呼び出し（サイズ優先最適化）” 選択時）

- **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。**

(d) **“ 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）**



- [HL + B] **を使用したオブジェクトを出力する（スタティック・モデルのみ）** [-qe](P)

  [HL + B] **をオペランドに使用したコード出力を行いたい場合に，チェック・ボックスをチェックします。**

- [HL] .bit **を使用したオブジェクトを出力する** [-qh](T)

  [HL] .bit **をオペランドに使用したコード出力を行いたい場合に，チェック・ボックスをチェックします。**

- **積極的な最適化を行う** (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。**

(5) ［ディバグ］タブ

図 3-14 〈コンパイラオプションの設定〉ダイアログ（［ディバグ］タブ選択時）

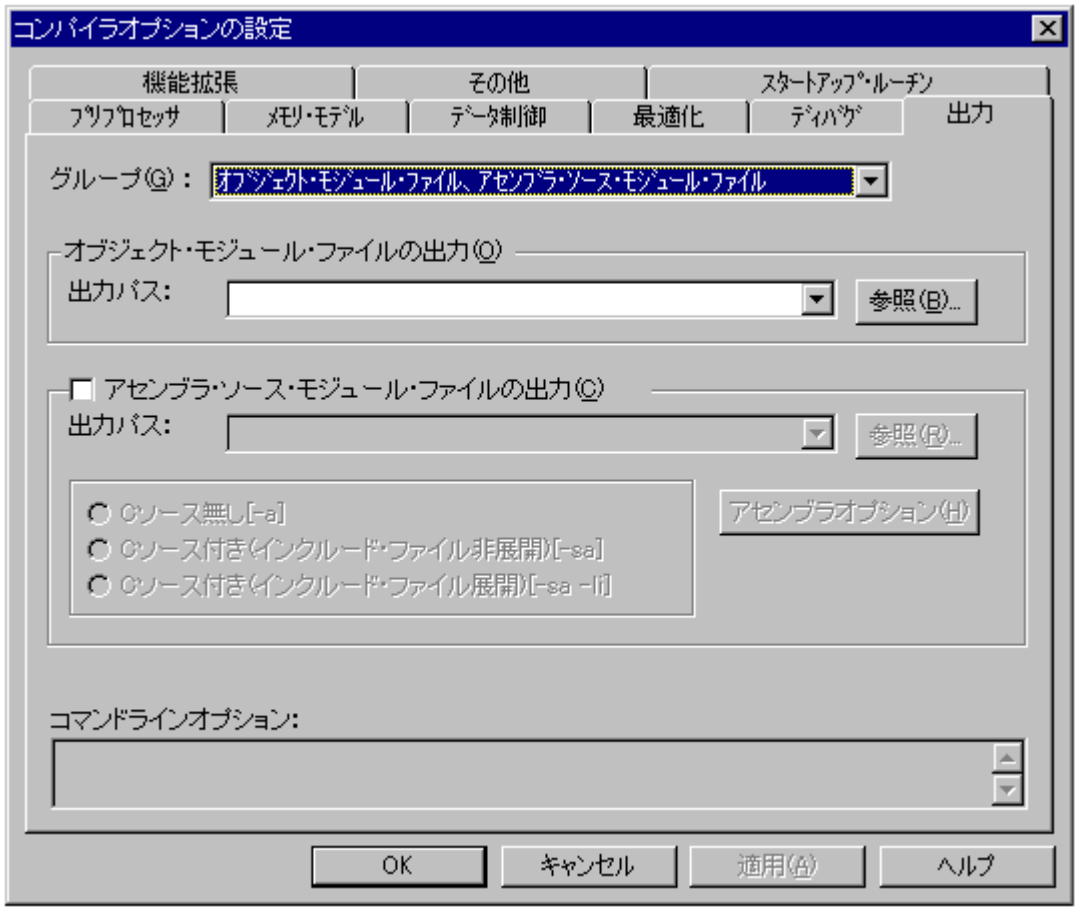
- **ディバグ情報の出力** (O)

  -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，ディバ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。**PM plus **のオプションで，「ディバグ」が無効の場合は，〈ディバグ〉設定ダイアログの設定はできなくなり，ディバグ情報を出力しません。**

(6) ［出力］タブ

(a) “ 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。［参照 (B)］ボタンによっても指定できます。

  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。［参照 (R)］ボタンによっても指定できます。

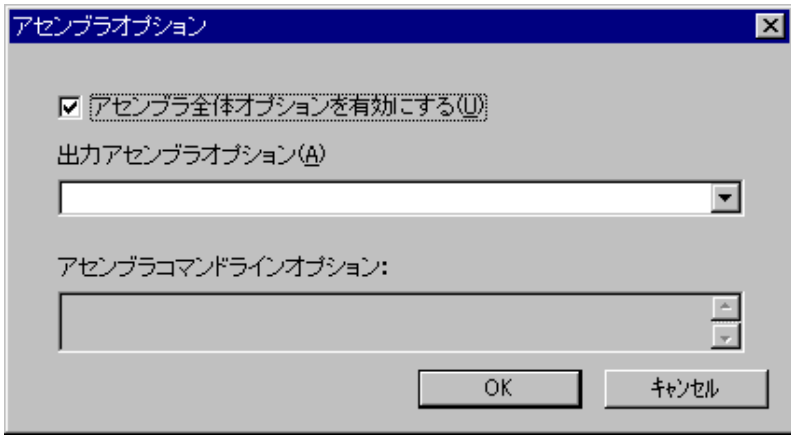
  全体オプション指定時は，常にパス名が指定されたものとして処理を行います。

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したい場合に，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)

コンパイラの出力アセンブラ・ソース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・ボックスの右側の＜ドロップダウンリスト＞により，過去に入力した履歴を選択できます。

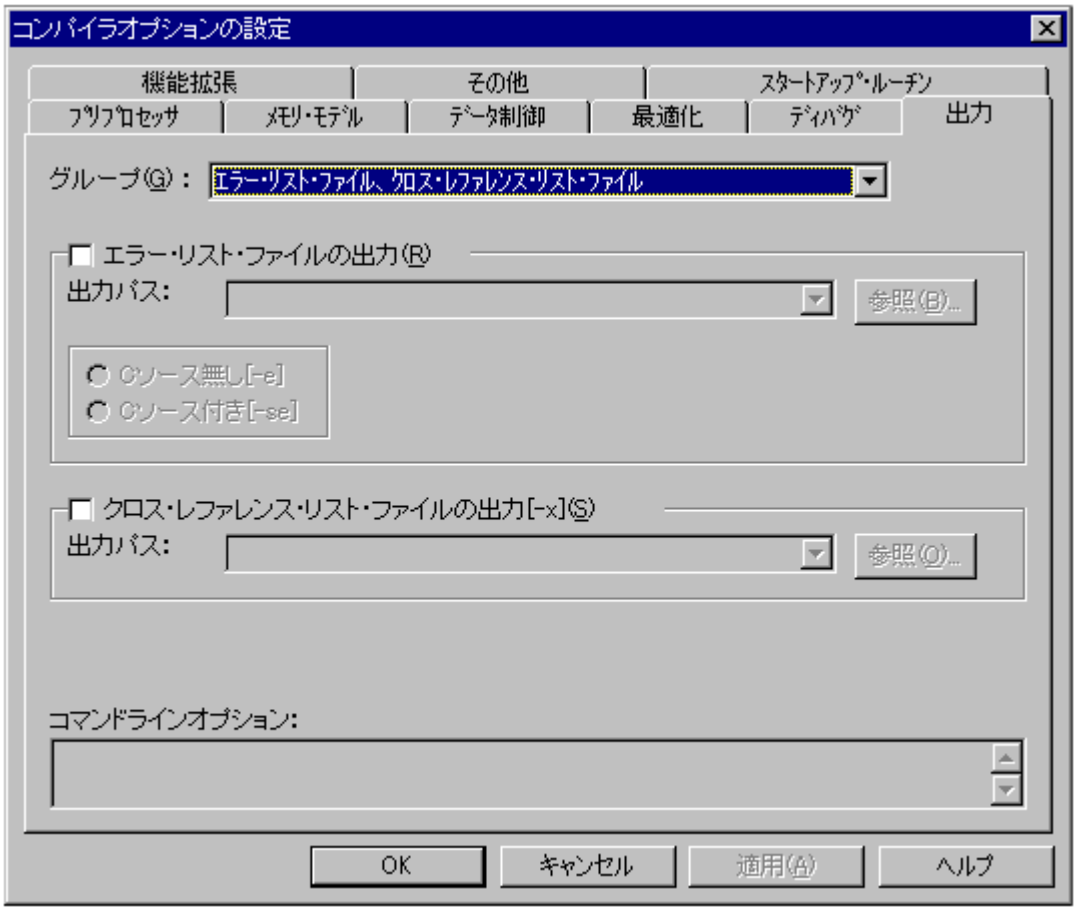
注意　デバイス種別指定（-C），デバイスファイル指定（-Y），およびパラメータ・ファイル指定（-F）は，ツール DLL で設定するため，記述しないでください。

- 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。
文字列がボックス内に収まりきらない場合，＜スクロールバー＞でスクロールします。
各ボタン，ボックス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。
オプション文字列には，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が表示されます。

(b) “ グループ (G) ” で “ エラー・リスト・ファイル、クロス・レファレンス・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・レファレンス・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。［参照 (B)**］ボタンによっても指定できます。**

  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・レファレンス・リスト・ファイル** [-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・レファレンス・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。［参照 (O)**］ボタンによっても指定できます。**

  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。

(c) “ 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](M)

  -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)

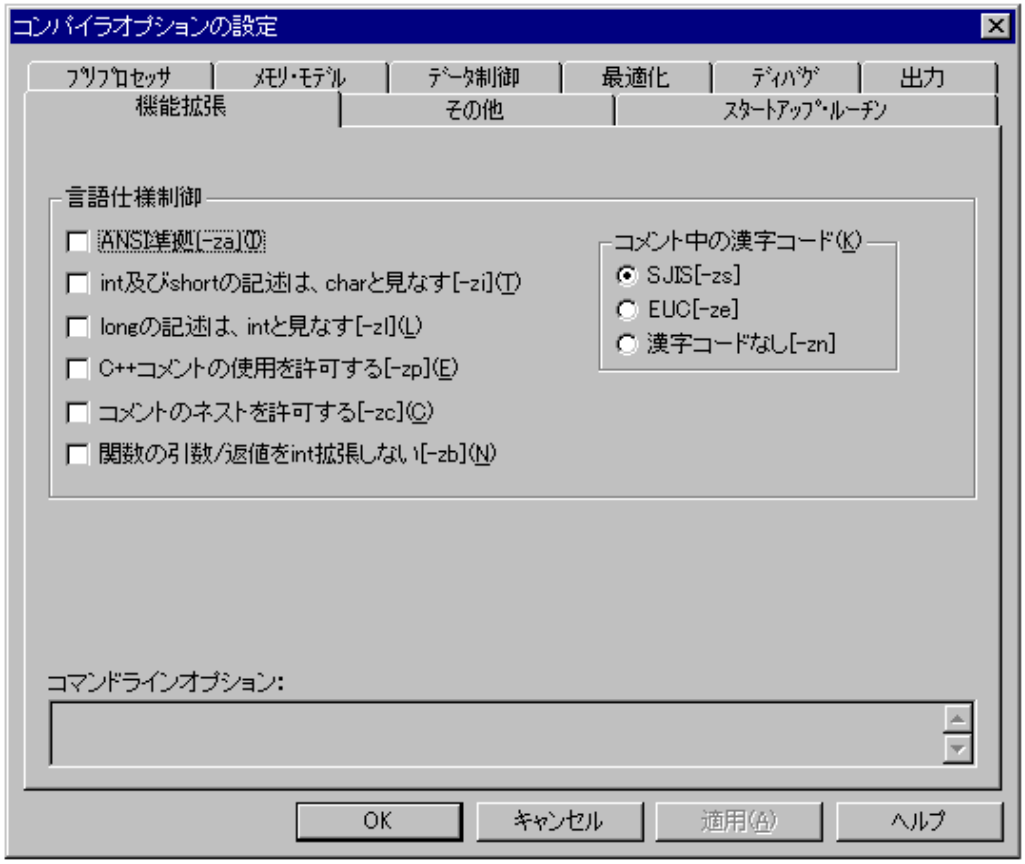
  -KN **オプションを有効にする場合にチェック・ボックスをチェックします。**

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。［参照 (B)］ボタンによっても指定できます。
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。ボックスに表示されている数値を増減する場合は，＜アップダウン＞ボタンで変更できます。
- 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。ボックスに表示されている数値を増減する場合は，＜アップダウン＞ボタンで変更できます。
- 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。ボックスに表示されている数値を増減する場合は，＜アップダウン＞ボタンで変更できます。

(7) ［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。**

int**，及び** short **の記述は、**char **と見なす** [-zi](T)

-ZI **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。**

long **の記述は、**int **と見なす** [-zl](L)

-ZL **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。**

スタティック・モデルでは，このオプションはディフォールトとなっています。

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)

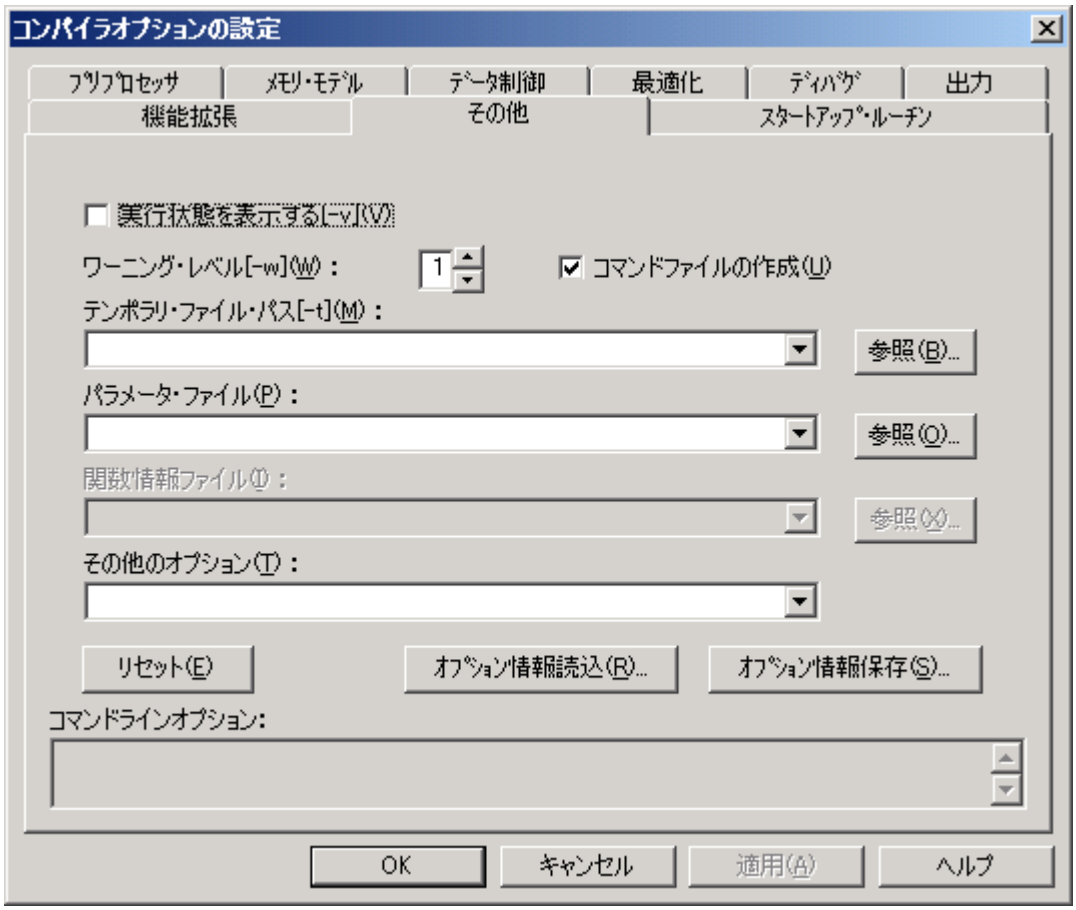
-ZB **オプションを有効にする場合にチェック・ボックスをチェックします。**

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。**

(8) **［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 **オプションのレベルを変更する場合は，＜アップダウン＞ボタンで変更します。**

- **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。このチェック・ボックスはディフォールトで選択されています。

- **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ディレクトリを，コンボ・ボックスに入力します。**

- **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。**

  コンボ・ボックスの右側の＜ドロップダウンリスト＞により，過去に入力した履歴を選択できます。

- **関数情報ファイル** [-mf](I)

  -mf **オプションで指定する関数情報ファイル名をコンボ・ボックスに入力します。**

- **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・ボックスの右側の＜ドロップダウンリスト＞により，過去に入力した履歴を選択できます。

- **［リセット** (E)**］ボタン**

  オプション設定をディフォールト状態に戻します。

- **［オプション情報読込** (R)**］ボタン**

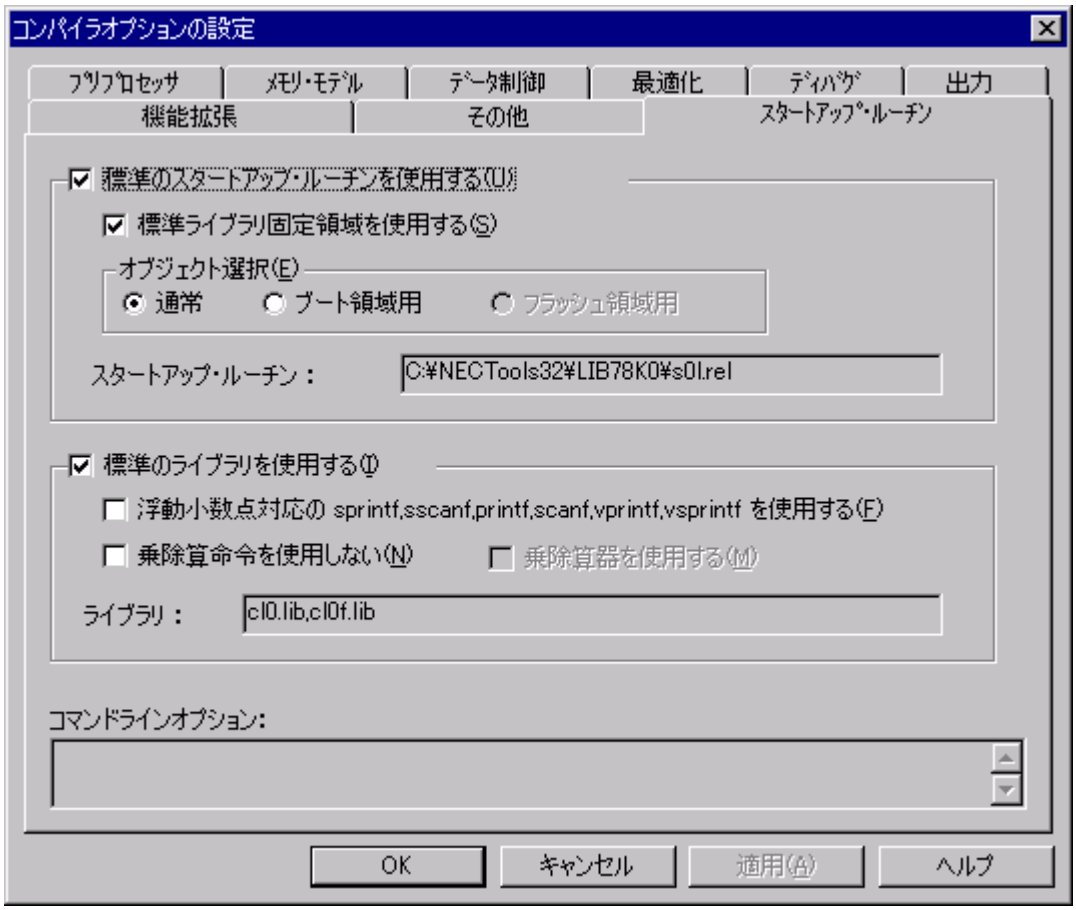
  オプション設定を保存したオプション情報ファイルを読み込みます。

- **［オプション情報保存** (S)**］ボタン**

  このボタンは，［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。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。**

(9) ［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）



ソース指定時は，〈スタートアップ・ルーチン〉設定ダイアログの設定はできません。

- 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。

  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 ” で “ フラッシュ用オブジェクトを出力する ” のチェック・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・ルーチンの選択ができ，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み選択できます。

  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。**

  “ メモリ・モデル ” で “ スタティック・モデル ”, “ 自動的にパスカル関数修飾子を付与する ” オプションを指定した場合，浮動小数点対応の sprintf, sscanf, printf, scanf, vprintf, vsprintf **関数は，使用できません。**

  乗除算命令を使用しない (N)

  乗除算命令を持たない品種を使用する場合にチェック・ボックスをチェックします。

  乗除算器を使用する (M)

  乗除算器を持つ品種で，乗除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　乗除算器を持たない品種の場合，選択できません。

  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 plus からのビルド

PM plus を使用してビルドする方法を説明します。

PM plus 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 plus 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，ディバグを連続した作業として行えるようになります。

(1) PM plus 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［NECTools32］メニューが作成され，その中に PM plus などが登録されています。

PM plus を起動するには，メニューから［PM plus］をクリックします。

(2) プロジェクトの作成

PM plus 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は「PM plus」のユーザーズ・マニュアルを参照してください。

(3) 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ィフォールト [-QCJLVW] から，サイズ優先 [-QX3] に変更した例を示します。

図 3-22 〈コンパイラオプションの設定〉ダイアログ（［最適化］タブ選択時）

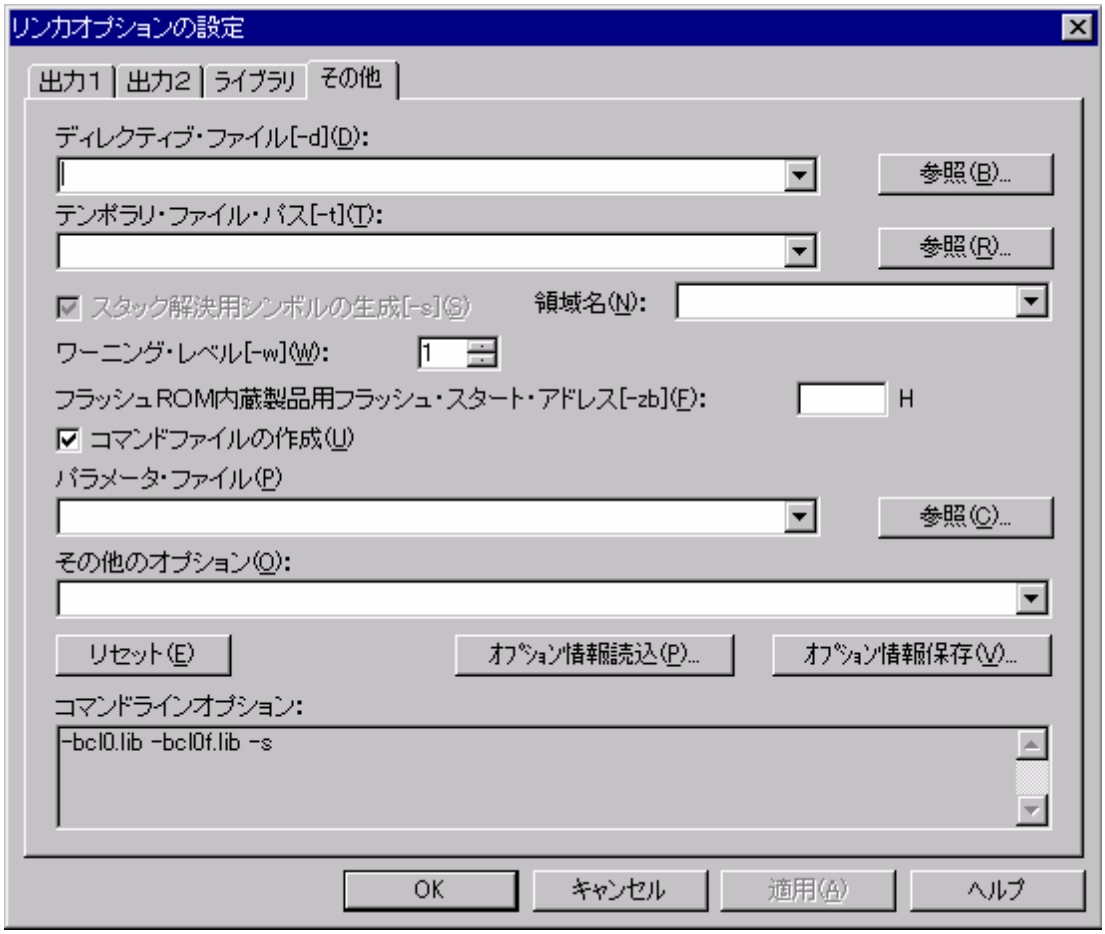
〈コンパイラオプションの設定〉ダイアログの［スタートアップ・ルーチン］タブで，標準のスタートアップ・ルーチンを使用する設定にしている場合は，コンパイラ標準のスタートアップ・ルーチンが，すべてのソースの前でリンクされます（〈リンカオプションの設定〉ダイアログに表示はされません）。

標準のライブラリを使用する設定にしている場合には，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。

図 3-23 〈リンカオプションの設定〉ダイアログ



(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 plus 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。

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

(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 ] cc78k0 [ △オプション ] △ C ソース名 [ △オプション ]
>[ パス名 ] ra78k0 [ △オプション ] △ アセンブラ・ソース名 [ △オプション ]
>[ パス名 ] lk78k0 [ △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，コンパイラ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，コンパイラ付属の浮動小数点用ライブラリ，コンパイラ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，コンパイラ付属のライブラリ，コンパイラ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，C コンパイラ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

(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)　C コンパイラ付属のライブラリ・ファイル（-B オプションで指定）

(iii)　C コンパイラ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)　C コンパイラ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)　C コンパイラ付属のライブラリ・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）

(i) C コンパイラ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

(ii) 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>cc78k0 -c054 s1.c -e -a -iC:\NECTools32\inc78k0 –yC:\NECTools32\dev
C>ra78k0 -c054 s2.asm -e -yC:\NECTools32\dev
C>lk78k0 s01.rel sl.rel s2.rel -bC:\NECTools32\lib78k0\cl0.lib -s
-osample.lmf -yC:\NECTools32\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「RA78K0 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

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ことがあります。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 ] CC78K0 △ -F パラメータ・ファイル名
>[ パス名 ] RA78K0 △ -F パラメータ・ファイル名
>[ パス名 ] LK78K0 △ -F パラメータ・ファイル名
```

次に使用例を示します。

```
C>cc78k0 -Fpara.pcc
C>ra78k0 -Fpara.pra
C>lk78k0 -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＞

```
-c054 sl.c -e -a -iC:\NECTools32\inc78k0 -yC:\NECTools32\dev
```

＜ para.pra の内容＞

```
-c054 s2.asm -e -yC:\NECTools32\dev
```

＜ para.plk の内容＞

```
s01.rel s1.rel s2.rel -bC:\NECTools32\lib78k0\cl0.lib -s -osample.lmf
-yC:\NECTools32\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「RA78K0 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，使用できます。

3.3.1 PM plus からのコンパイルからリンク

PM plus **を使用して** MAKE **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) **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**

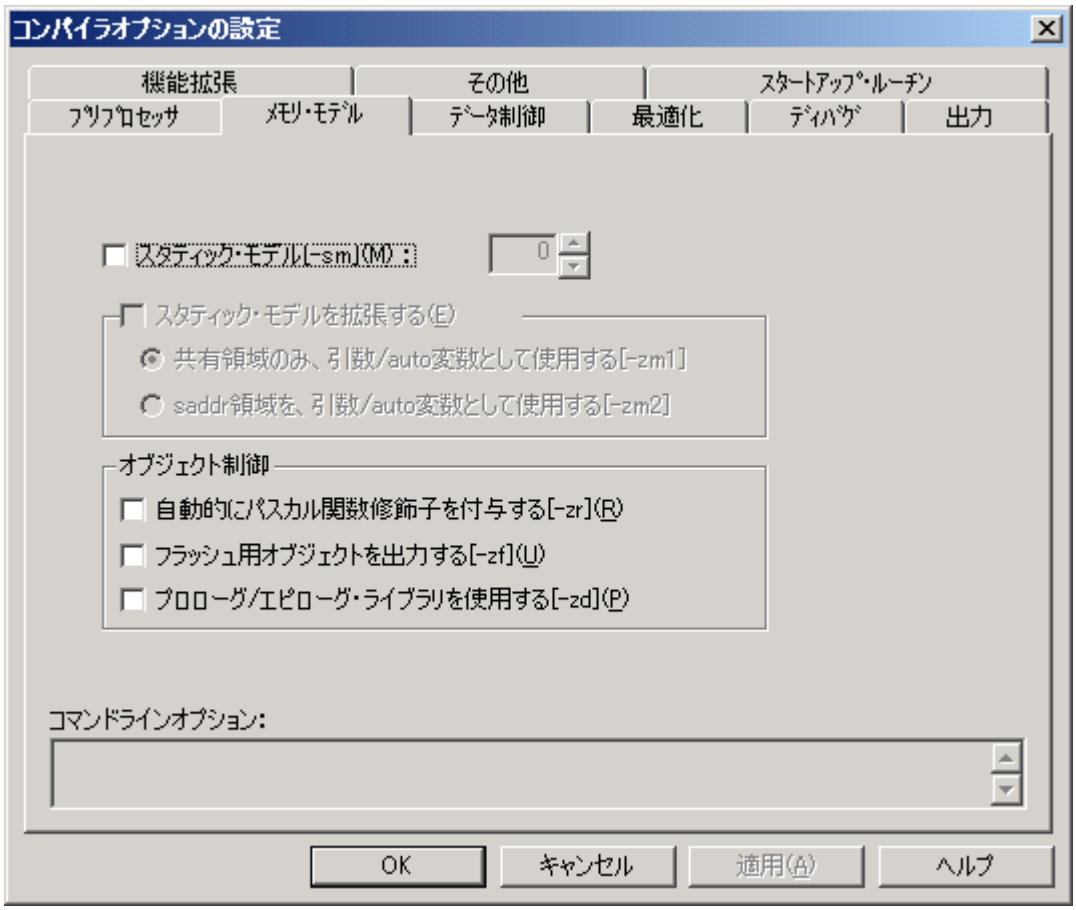
プロジェクト作成の終了時に自動的に作成されるメイク・ファイルは，ビルドのために必要最低限のオプションのみが設定されています。プロジェクト特有のオプションは，［ツール (T)**］メニューで設定します。**

［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) **コンパイラ・オプションの設定**

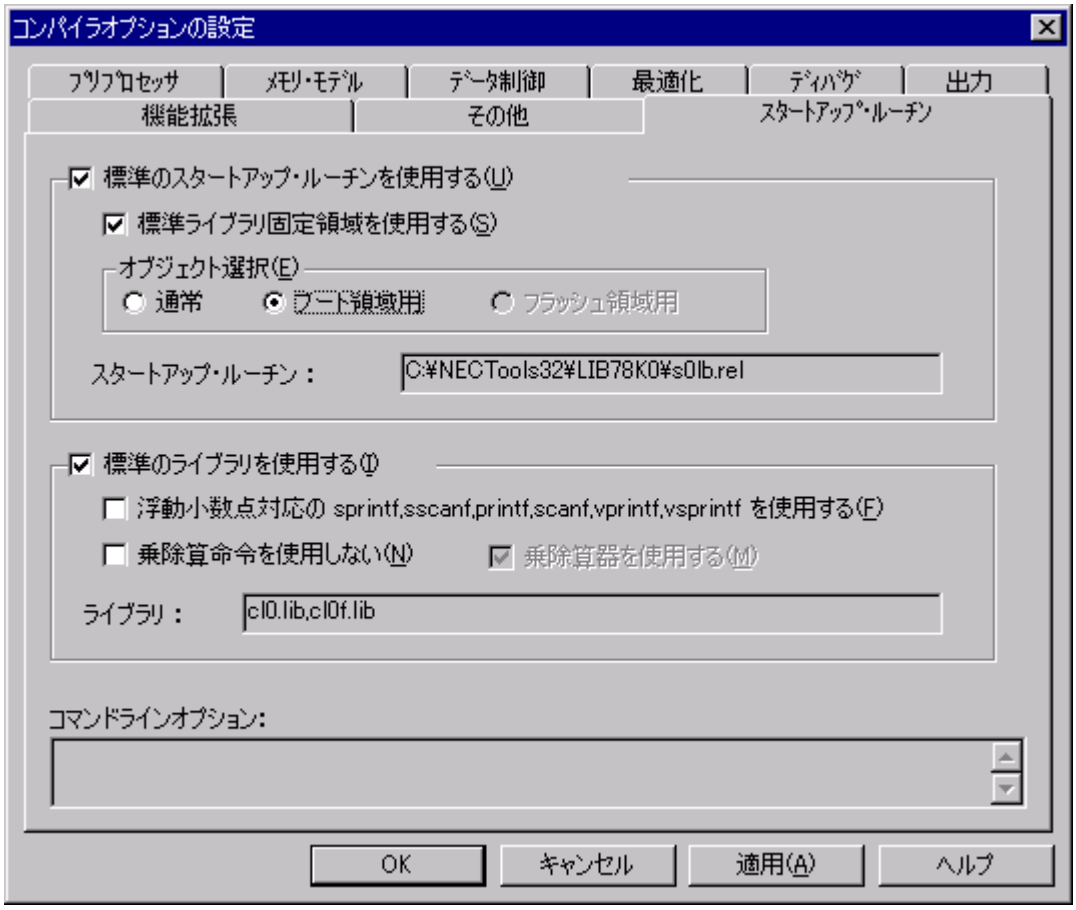
［メモリ・モデル］タブで，-ZF **オプションは指定しないでください。**

図 3-24 〈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］タブの“オブジェクト選択(E)”でブート領域用を選択してください。

図 3-25 ブート領域用の選択



(ii) **リンカ・オプションの設定**

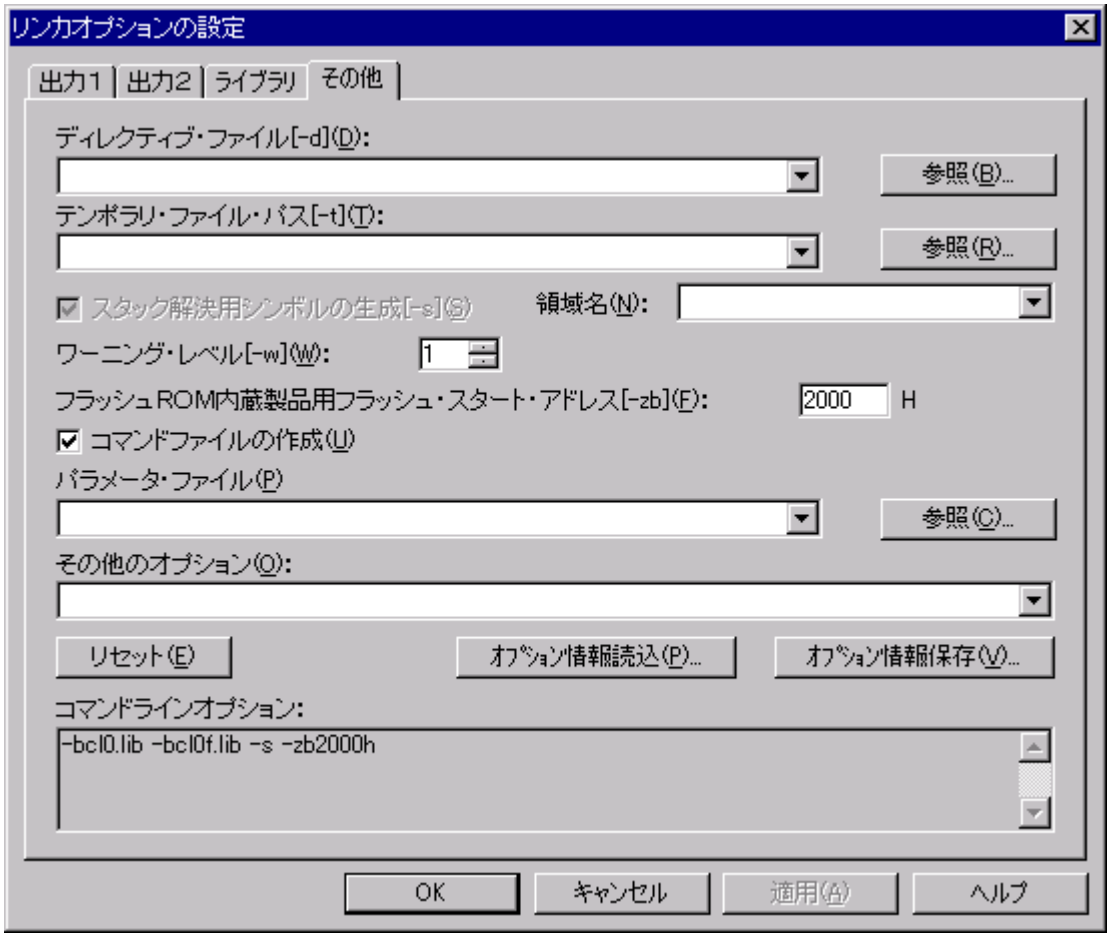
“ フラッシュ ROM **内蔵製品用フラッシュ・スタート・アドレス** [-zb](E) **” を指定し，**［OK］**ボタンを押します。**

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) **” 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，「RA78K0 **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。**

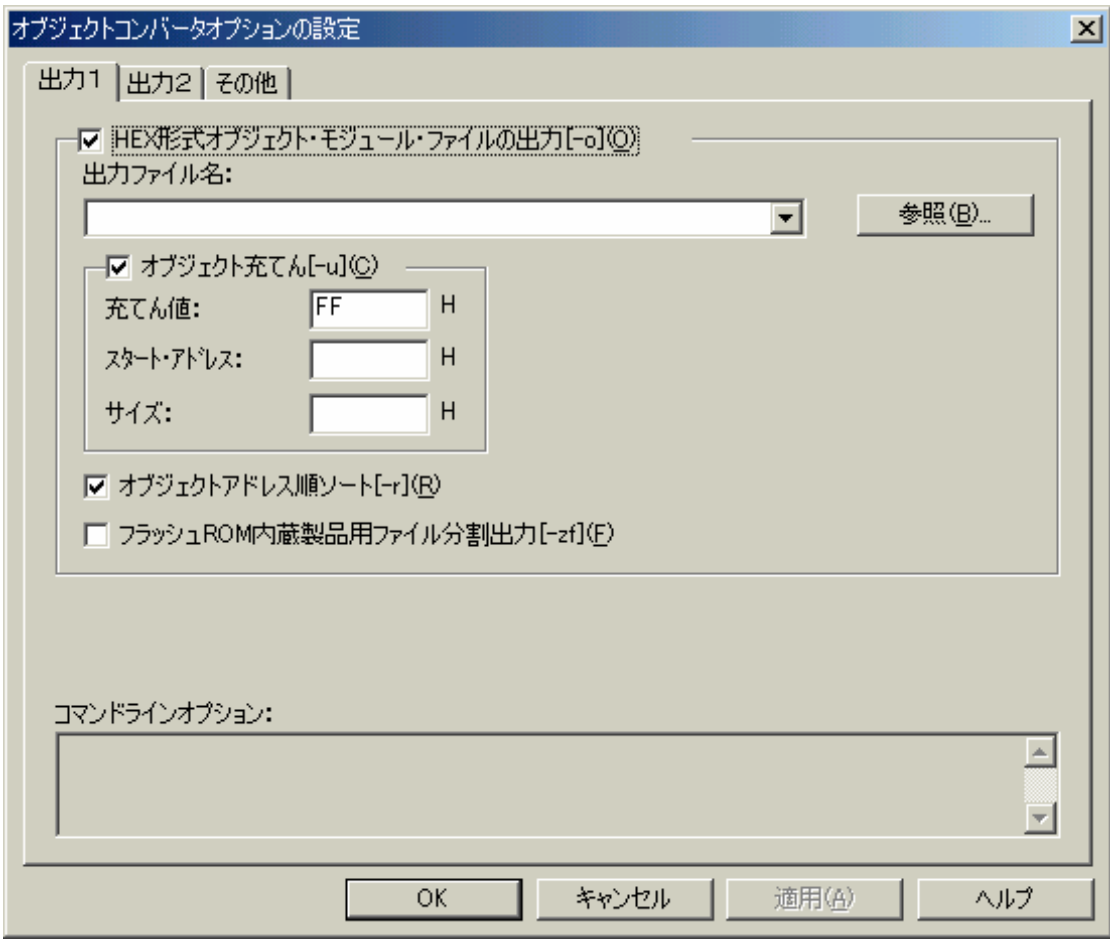
図 3-26 **〈リンカオプションの設定〉ダイアログ**



(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) **” を指定しないでください。**

図 3-27 **〈オブジェクトコンバータオプションの設定〉ダイアログ**



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 plus **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) 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

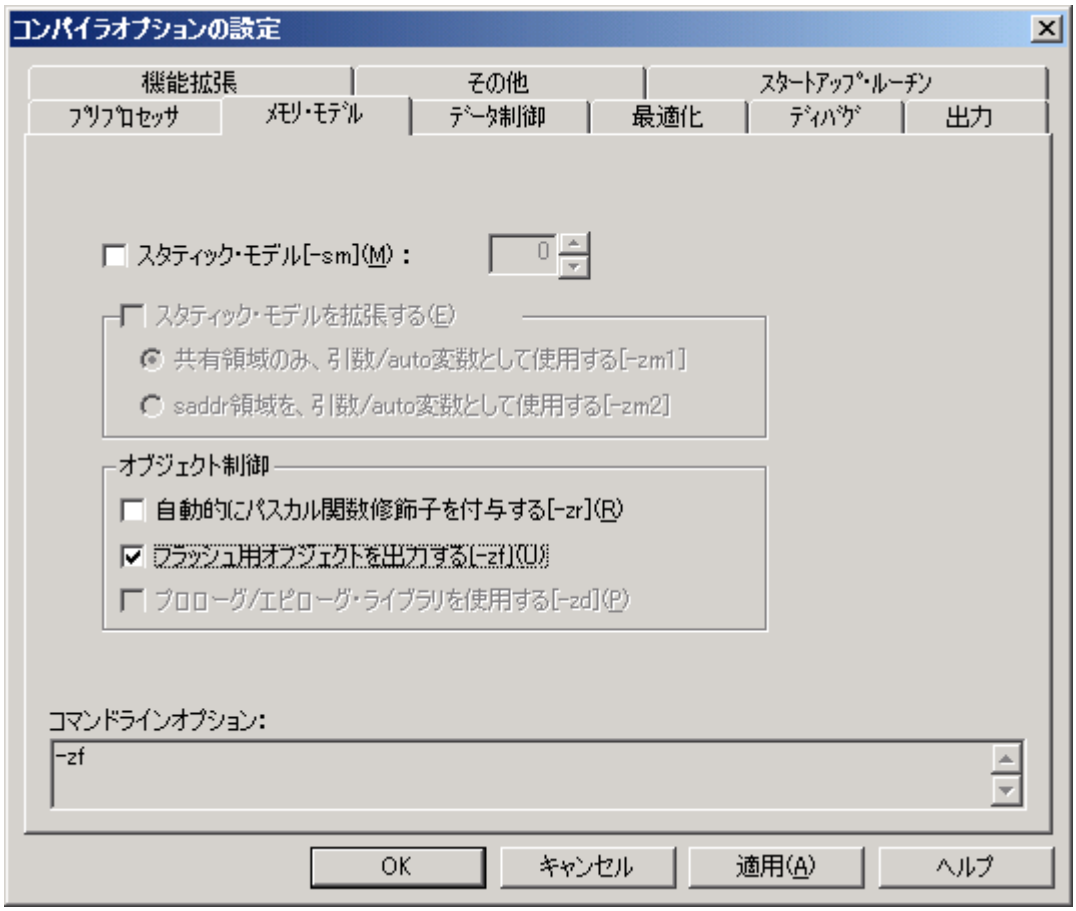
プロジェクト作成の終了時に自動的に作成されるメイク・ファイルは，ビルドのために必要最低限のオプションのみが設定されています。プロジェクト特有のオプションは，［ツール (T)］メニューで設定します。

［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) ” を指定してください。

図 3-28 〈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］タブのオブジェクト選択で，フラッシュ領域用が自動選択されます。

(ii) リンカ・オプションの設定

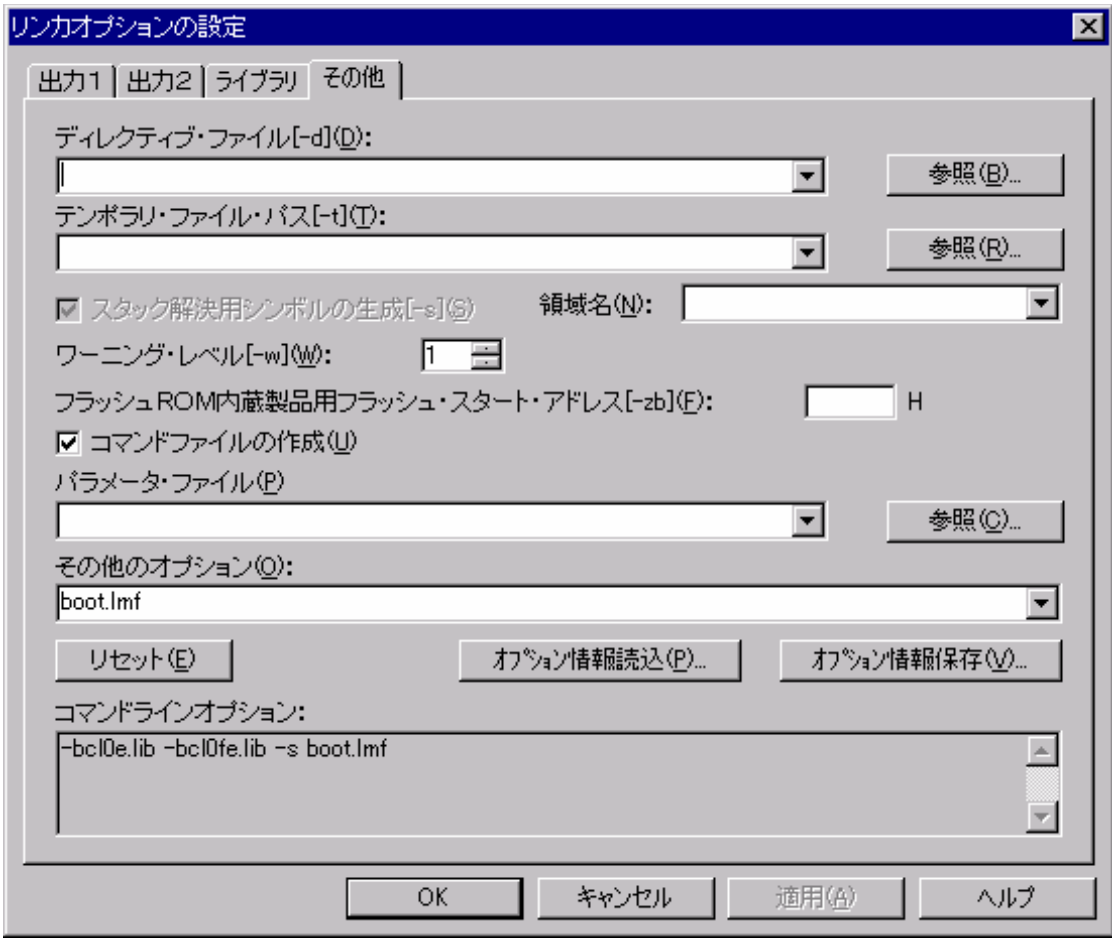
リンクするブート領域用ロード・モジュール・ファイルを指定し，［OK］ボタンを押します。

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) ” が自動的に設定されます。
“ その他のオプション (O) ” に作成したブート領域用ロード・モジュール・ファイルを指定してください。

備考　リンカ・オプションについては，「RA78K0 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。

図 3-29 〈リンカオプションの設定〉ダイアログ



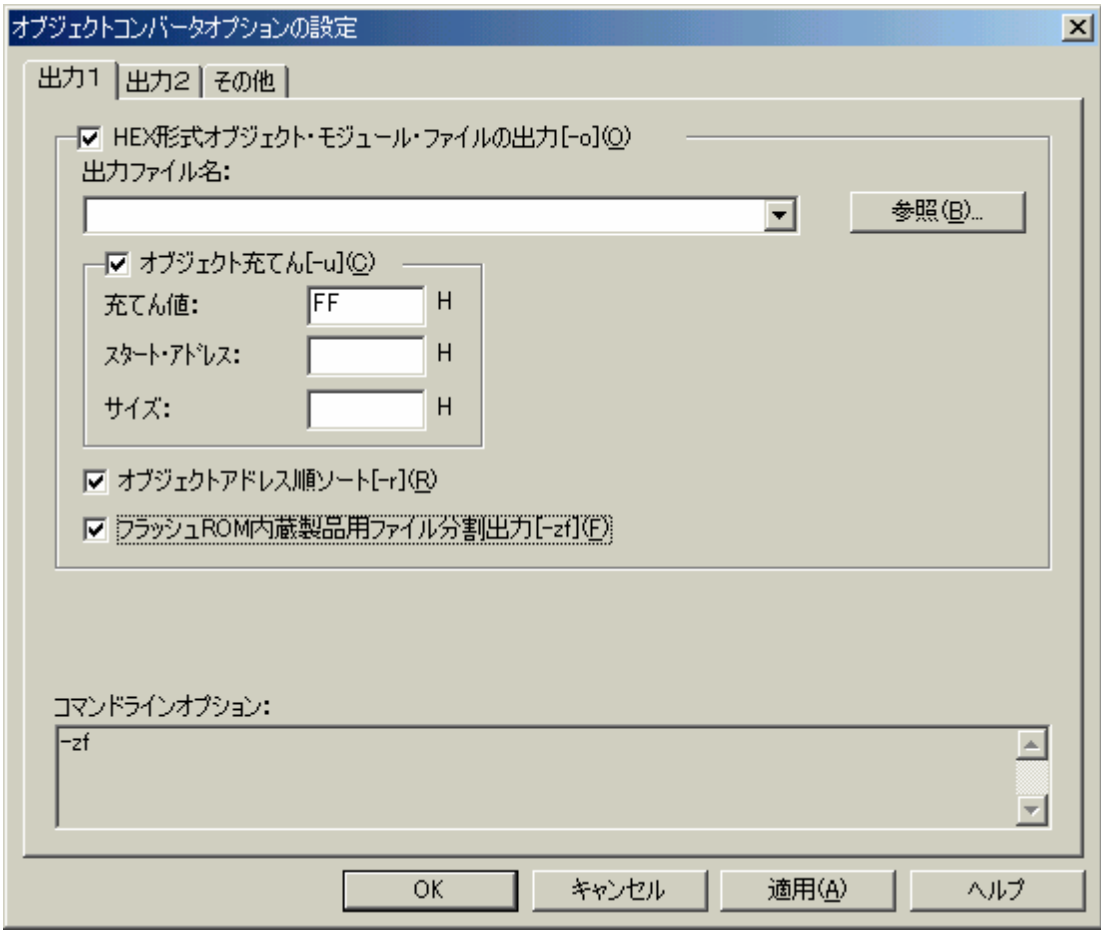
(iii) オブジェクト・コンバータ・オプションの設定（フラッシュ領域用）

“ フラッシュ ROM 内蔵製品用ファイル分割出力 [-zf](F) ” を必ず指定してください。

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。

図 3-30 〈オブジェクトコンバータオプションの設定〉ダイアログ



(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)**］→［ビルド** (B)**］メニューから，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 plus **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（DOS 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 ] cc78k0 [ △オプション ] △ C ソース名 [ △オプション ]
>[ パス名 ] ra78k0 [ △オプション ] △アセンブラ・ソース名 [ △オプション ]
>[ パス名 ] lk78k0 [ △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>cc78k0 -cf0078 boot.c -iC:\NECTools32\inc78k0 -yC:\NECTools32\dev
```

例 2 **ブート領域用プログラムのリンク**

```
C>lk78k0 s01b.rel boot.rel -bC:\NECTools32\lib78k0\cl0.lib -s -oboot.lmf
    -zb2000h -yC:\NECTools32\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>oc78k0 boot.lmf -u0FFh -oboot.lmf -yC:\NECTools32\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 4 **フラッシュ領域用プログラムのコンパイル**

```
C>cc78k0 -cf0078 flash.c -zf -iC:\NECTools32\inc78k0
    -yC:\NECTools32\dev
```

例 5 **フラッシュ領域用プログラムのリンク**

```
C>lk78k0 boot.lmf s0le.rel flash.rel -bC:\NECTools32\lib78k0\cl0e.lib
    -s -oflash.lmf -yC:\NECTools32\dev
```

例 6 **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>oc78k0 flash.lmf -u0FFh -r -oflash.lmf -yC:\NECTools32\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「RA78K0 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，コンパイラ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) C コンパイラ付属のライブラリ・ファイル（-B オプションで指定）
(iii) C コンパイラ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) C コンパイラ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) C コンパイラ付属のライブラリ・ファイル（-B オプションで指定）

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) C コンパイラ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**

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ことがあります。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 ] cc78k0 △ -F パラメータ・ファイル名
> [ パス名 ] ra78k0 △ -F パラメータ・ファイル名
> [ パス名 ] lk78k0 △ -F パラメータ・ファイル名
```

次に使用例を示します。

```
C>cc78k0 -Fpara.pcc
C>lk78k0 -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**＞

```
-cf0078 boot.c -iC:\NECTools32\inc78k0 -yC:\NECTools32\dev
```

＜ para.plk **の内容**＞

```
s0lb.rel boot.rel -bC:\NECTools32\lib78k0\cl0.lib -s -oboot.lmf -zb2000h
-yC:\NECTools32\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「**RA78K0 **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 手順（バンク関数使用時）

この機能は，バンク機能を持つデバイスにのみ，使用できます。

3.4.1 PM plus からのコンパイルからリンク

PM plus を使用して MAKE 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) 関数情報ファイルの生成

(a) プロジェクトの作成

特別な設定は必要ありません。

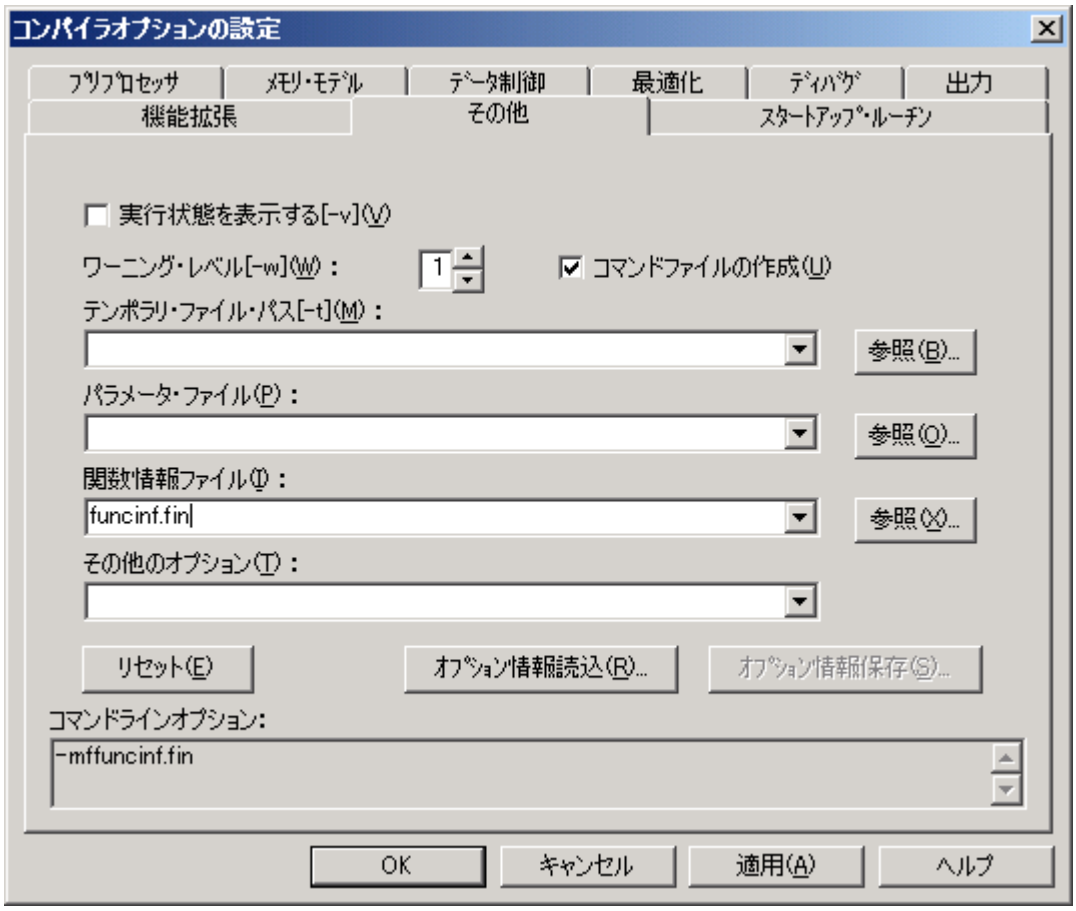
プロジェクトを作成し，ソース・ファイルを登録してください。

(b) コンパイラのオプション設定

［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。

［その他］タブで “ 関数情報ファイル (I) ” に関数情報ファイル名を指定してください。

図 3-31 関数情報ファイル名の指定



指定した関数情報ファイルは，PM plus のプロジェクト関連ファイルに登録されます。

(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 plus **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。

コンパイルが正常終了することを確認してください。

コンパイルが正常終了すると，関数情報ファイルが生成されます。

この時点での関数情報ファイルは，全て共通領域に割り当てる設定になっているため，リンクで異常終了する場合があります。

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) **関数情報ファイルの編集**

ビルド終了時に出力されるリンカのエラー・メッセージの内容を確認し，関数情報ファイルの編集を行います。

(a) **正常終了した場合**

バンク機能を使用せずに，全てのプログラムを割り当てることができました。

バンク機能を使用する必要はありません。

(b) **共通領域に割り当てができない場合**

リンカで以下のエラー・メッセージが出力されます。

RA78K0 error E3206: Segment ' @@CODE ' can't allocate to memory - ignored.

関数情報ファイルを編集し，バンク領域への配置を行ってください。

```
/ #0xxxx
// 78K/0 Series C Compiler Vx.xx Function Information File

main.c   := C       (30000)
{
            func1 ;
            func2 ;
}

sub1.c   := C       (10000)
{
            func3 ;
            func4 ;
}

sub2.c   := C       (10000)
{
            func5 ;
}

// *** Code Size Information ***
// COMMON       : 50000 byte
// BANK00       :       0 bytes
// BANK01       :       0 bytes
// BANK02       :       0 bytes
// BANK03       :       0 bytes
```

sub1.c をバンク 0 へ，sub2.c をバンク 1 に配置先を変更します。

```
/ #0xxxx
// 78K/0 Series C Compiler Vx.xx Function Information File

main.c   := C       (30000)
{
        func1 ;
        func2 ;
}

sub1.c   := 0       (10000)
{
        func3 ;
        func4 ;
}

sub2.c   := 1       (10000)
{
        func5 ;
}

// *** Code Size Information ***
// COMMON       : 50000 byte
// BANK00       :       0 bytes
// BANK01       :       0 bytes
// BANK02       :       0 bytes
// BANK03       :       0 bytes
```

注意　配置先の変更以外の修正を行わないでください。コード・サイズの情報は，ビルド時に更新されます。

ただし，バンク領域に配置したアセンブリ言語ルーチンを C 言語から呼び出す場合は，配置情報を追記する必要があります。

追記した配置情報のコード・サイズの情報は，更新されません。

(c) **バンク領域に割り当てができない場合**

リンカで以下のエラー・メッセージが出力されます。

RA78K0 error E3206: Segment ' @@BANK(n) ' can't allocate to memory - ignored.

バンク番号 n にプログラムを割り当てることができませんでした。関数情報ファイルを編集し，共通領域，または他のバンク領域への配置を行ってください。

```
/ #0xxxx
// 78K/0 Series C Compiler Vx.xx Function Information File

main.c   := C       (20000)
{
            func1 ;
            func2 ;
}

sub1.c   := 0       (10000)
{
            func3 ;
            func4 ;
}

sub2.c   := 0       (10000)
{
            func5 ;
}

// *** Code Size Information ***
// COMMON       : 20000 byte
// BANK00       : 20000 bytes
// BANK01       :       0 bytes
// BANK02       :       0 bytes
// BANK03       :       0 bytes
```

sub2.c をバンク 1 に配置先を変更します。

```
/ #0xxxx
// 78K/0 Series C Compiler Vx.xx Function Information File

main.c   := C       (20000)
{
          func1 ;
          func2 ;
}

sub1.c   := 0       (10000)
{
          func3 ;
          func4 ;
}

sub2.c   := 1       (10000)
{
          func5 ;
}

// *** Code Size Information ***
// COMMON       : 20000 byte
// BANK00       : 20000 bytes
// BANK01       :       0 bytes
// BANK02       :       0 bytes
// BANK03       :       0 bytes
```

注意　配置先の変更以外の修正を行わないでください。コード・サイズの情報は，ビルド時に更新されます。

ただし，バンク領域に配置したアセンブリ言語ルーチンを C 言語から呼び出す場合は，配置情報を追記する必要があります。

追記した配置情報のコード・サイズの情報は，更新されません。

(d) **関数情報ファイルの更新があった場合**

リンカで以下のエラー・メッセージが出力されます。

RA78K0 error E3403: Symbol ' **関数名** ' unmatched type in file ' **モジュール名** '.

First defined in file ' **モジュール名** '

ソース・ファイルの追加，削除，または関数の追加，削除を行った場合，コンパイラは，関数情報ファイルの更新を行いますが，ビルドの途中に関数情報ファイルの変更があった場合，更新前にコンパイルしたソース・ファイルに関数情報ファイルの内容が反映されません。

全てのソース・ファイルに変更内容を反映するため，再度ビルドを行ってください。

(3) **関数情報ファイルの更新**

関数情報ファイルの編集後，ビルドを行います。

ビルドを行うことで，関数情報ファイルの配置先の指定に従い配置を行います。

ビルドが終了すると，ソース・ファイル，関数の情報，および配置先の指定に従い配置を行った場合のコード・サイズが反映された関数情報ファイルが生成されます。

再度，リンカでエラーが出力された場合は，「(2) 関数情報ファイルの編集」の作業を繰り返してください。

(4) **関数情報ファイルの編集の注意点**

- **関数情報ファイルのフォーマットについては，「CC78K0 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。**
- **コード・サイズは，関数の配置場所により変化します。また，呼び出す先の関数の配置場所によっても変化します。**
- **関数情報ファイルのコード・サイズ情報には，ライブラリ，スタートアップ・ルーチンのサイズは含まれません。詳細なコード・サイズは，リンカが出力するリンク・リスト・ファイルを参照してください。リンク・リスト・ファイルについては，「RA78K0 アセンブラ・パッケージ 操作編」を参照してください。**
- **バンクのサイズよりも大きいファイルをバンクに配置することはできません。共通領域に配置するか，ソースを分割してください。**

3.4.2 コマンド行（DOS プロンプト）でのコンパイルからリンク

(1) 関数情報ファイルの生成

次にバンク機能を使用する場合の例を示します。

```
C>cc78k0 -cf054780 file1.c -mffuncinf.fin -iC:\NECTools32\inc78k0 -yC:\NECTools32\dev
C>cc78k0 -cf054780 file2.c -mffuncinf.fin -iC:\NECTools32\inc78k0 -yC:\NECTools32\dev
C>lk78k0 s0l.rel file1.rel file2.rel -bC:\ECTools32\lib78k0\cl0.lib -s -osample.lmf -yC:\NECTools32\dev
```

コンパイルが正常終了することを確認してください。

コンパイルが正常終了すると，関数情報ファイルが生成されます。

この時点での関数情報ファイルは，全て共通領域に割り当てる設定になっているため，リンクで異常終了する場合があります。

(2) 関数情報ファイルの編集

編集方法は「3.4.1 PM plus からのコンパイルからリンク」を参照してください。

(3) 関数情報ファイルの更新

関数情報ファイルの編集後，再度コンパイル，リンクを行います。

```
C>cc78k0 -cf054780 file1.c -mffuncinf.fin -iC:\NECTools32\inc78k0 -yC:\NECTools32\dev
C>cc78k0 -cf054780 file2.c -mffuncinf.fin -iC:\NECTools32\inc78k0 -yC:\NECTools32\dev
C>lk78k0 s0l.rel file1.rel file2.rel -bC:\ECTools32\lib78k0\cl0.lib -s -osample.lmf -yC:\NECTools32\dev
```

コンパイル，リンクを行うことで，関数情報ファイルの配置先の指定に従い配置を行います。

リンクが終了すると，ソース・ファイル，関数の情報，および配置先の指定に従い配置を行った場合のコード・サイズが反映された関数情報ファイルが生成されます。

再度，リンカでエラーが出力された場合は，「(2) 関数情報ファイルの編集」の作業を繰り返してください。

3.5 C コンパイラの入出力ファイル

CC78K0 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，クロスレファレンス・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。

CC78K0 の入出力ファイルを次に示します。

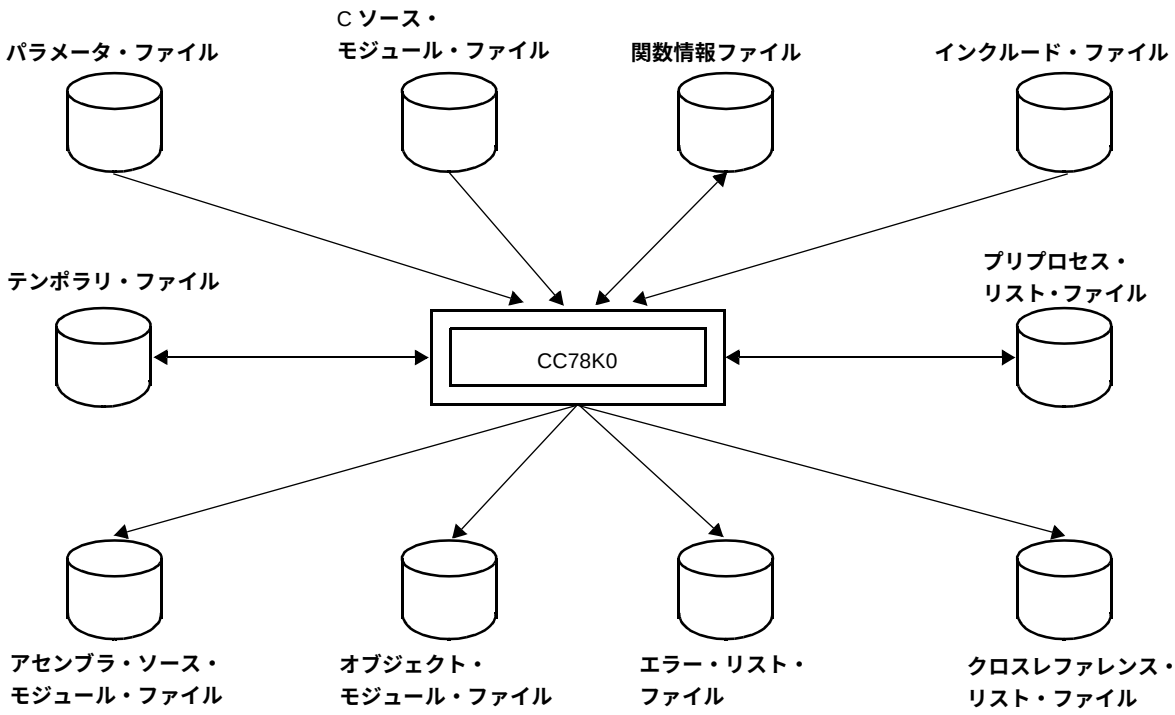
表 3-1 C コンパイラの入出力ファイル

種類	ファイル名	説明	ディフォールト・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・ファイルです。 - ユーザ作成ファイルです。	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です。 - C 言語で記述されたファイルです。 - ユーザ作成ファイルです。	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です。	pcc
	関数情報ファイル	- 関数の配置先を指定するファイルです。	fin
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・ファイルです。	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・ファイルです。	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・ファイルです。 - ASCII イメージ・ファイルです。	ppl
	クロスレファレンス・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・ファイルです。	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・ファイルです。	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です。 - 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。

- 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 3-32 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，クロスレファレンス・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。

3.6 実行開始メッセージ，終了メッセージ

3.6.1 実行開始メッセージ

CC78K0 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。

```
78K/0 Series C Compiler Vx.xx  [ xx xxx xxxx ]
    Copyright ( C ) NEC Electronics Corporation xxxx , xxxx
```

3.6.2 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，コンパイラは，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 : uPD780xx
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，コンパイラは，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) : CC78K0 warning W0745 : Expected function prototype
PRIME.C(20) : CC78K0 warning W0745 : Expected function prototype
PRIME.C(26) : CC78K0 warning W0622 : No return value
PRIME.C(37) : CC78K0 warning W0622 : No return value
PRIME.C(44) : CC78K0 warning W0622 : No return value

Target chip : uPD780xx
Device file : Vx.xx

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

コンパイル中にコンパイラ処理継続が不可能な致命的エラーが検出された場合，コンパイラ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。

```
  C>cc78k0 -c054 -e prime.c –m

  78K/0 Series C Compiler Vx.xx  [ xx xxx xxxx ]
     Copyright ( C ) NEC Electronics Corporation xxxx , xxxx

CC78K0 error F0018 : Option is not recognized ‘ -m ’
Please enter ‘ CC78K0  -- ‘ , 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。

コンパイラがエラー・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　CC78K0 の機能

この章では，CC78K0 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

CC78K0 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表 4-1 で示します。

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，または xor a , a（デバイスによって異なります）
	(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。

表 4-1 最適化手法

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 ;
			{ for (i = 0 ; i < 10 ; i++)
			： → {
			a = b + c ; ：
			： ：
			} }

	(10)	無用命令の削除	a = a ; → 削除 a = b ; のあと，a が参照されません。 → 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 が参照されません（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 / -RK / -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) 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。

(8) ～ (13) の最適化は，将来サポート予定のものです。

(14)，(15) は，最適化オプションの指定によらず行われます。

(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。

CC78K0 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ます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。

(1) s0*.rel

スタートアップ・ルーチン（ROM 化対応）。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 というレーベル（シンボル）が付けられます。

(2) cl0*.lib

CC78K0 に添付されているライブラリ。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。

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 cstart には先頭に _@ が付加されます。

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。

(3) *.lib

ユーザ作成のライブラリ。シンボルの先頭に _ が付加されます。

注意　CC78K0 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「2.6.4 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，CC78K0 のコンパイラ・オプションについて説明します。

C コンパイラを起動する際に，コンパイラ・オプションを指定できます。コンパイラ・オプションは，コンパイラ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えます。

5.1　指定方法

コンパイラ・オプションは，次の方法で指定できます。

(1) C コンパイラを起動するときにコマンド行で指定します。

(2) PM plus 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。

(3) 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。

例

```
cc78k0 △ -c054 △ prime.c △ -aprime.asm △ -qx3
```

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	-G	-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。

【△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。

【○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です。

例 1

```
C>cc78k0  -c054  -e  sample.c  -no  -rd  -g
```

備考　-RD, -G オプションは，無効となります。

例 2

```
C>cc78k0  -c054  -e  sample.c  -p  -k
```

備考　-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。

例 3

```
C>cc78k0  -c054  -e  sample.c  -utest  -dtest = 1
```

備考　-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 4

```
C>cc78k0  -c054  -e  sample.c  -o  -no
```

備考　-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 plus 上では指定できません。PM plus 上でヘルプを参照する場合は，PM plus 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 19 種類のオプションに分類できます。

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	
最適化指定	-Q	最適化種別を指定します。
	-NQ	
ディバグ情報出力指定	-G	C ソース・レベルのディバ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ディレクトリから読み込みます。
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	
クロスレファレンス・リスト・ファイル作成指定	-X	クロスレファレンス・リスト・ファイルの出力を指定します。
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レベルを指定します。

表 5-2 コンパイラ・オプション一覧

分類	オプション	説明
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へ出力を指定します。
	-NV	
パラメータ・ファイル指定	-F	入力ファイル名，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・ファイル作成ディレクトリ指定	-T	テンポラリ・ファイルを指定したドライブ，ディレクトリに作成します。
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。
スタティック・モデル指定	-SM	オブジェクトのスタティック・モデルかノーマル・モデルを指定します。
関数情報ファイル指定	-MF	64K バイトを越えるコード部への配置をファイルから指定することを指示します。

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 CC78K0 を起動した場合の例です。PM plus 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>cc78k0  -c054  prime.c  -g
```

例 （PM plus 使用時）

図 5-1 〈コンパイラオプションの設定〉ダイアログ

(1) デバイス種別指定

デバイス種別指定（-C）

【記述形式】

-C デバイス種別

- 省略時解釈

  なし

【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。C コンパイラは，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 オプションで指定できるターゲット・デバイスと，それに対応するデバイス種別に関しては，デバイス・ファイルの製品添付資料 「使用上の留意点」を参照してください。
- CC78K0 使用時には，デバイス・ファイルが必要となります。

【注意】

- -C オプションは，省略できません。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を優先します。
- PM plus 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・オプションでは指定する必要はありません。

【使用例】

- コマンド行で指定します。ターゲット・デバイスは uPD78054 です。

C>cc78k0　-c054　prime.c

- C ソース中で指定して，起動させます。

```
#pragma         pc ( 054 )
#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・デバイス指定は，コマンド行で省略できます。

```
C>cc78k0  prime.c
```

- C ソース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 ( 054 )
#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>cc78k0  -c014  prime.c
```

コマンドライン実行後，次のようにコンパイラが実行されます。

```
78K/0 Series C Compiler Vx.xx  [ xx xxx xxxx ]
   Copyright ( C ) NEC Electronics Corporation xxxx , xxxx

sample\prime.c ( 1 ) : CC78K0 warning W0832 : Duplicated chip specifier
sample\prime.c ( 18 ) : CC78K0 warning W0745 : Expected function prototype
sample\prime.c ( 20 ) : CC78K0 warning W0745 : Expected function prototype
sample\prime.c ( 26 ) : CC78K0 warning W0622 : No return value
sample\prime.c ( 37 ) : CC78K0 warning W0622 : No return value
sample\prime.c ( 44 ) : CC78K0 warning W0622 : No return value

Target chip : uPD78014
Device file : Vx.xx

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

コマンド行で指定したターゲット・デバイス指定が優先されます。

(2) オブジェクト・モジュール・ファイル作成指定

オブジェクト・モジュール・ファイル作成指定（-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 plus 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 ” のコンボ・ボックスに出力先を指定してください。
- 個別オプション指定時は，出力ファイル名の変更もできます。
- ［出力］タブの “ 出力ファイル ” のコンボ・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】

- -NO と -O の両オプションを指定します（-O が優先されます）。

```
C>cc78k0  -c054  prime.c  -no  -o
```

(3) メモリ配置指定

メモリ配置指定（-R/-NR, -RB/-NR, -RD/-NR, -RK/-NR, -RS/-NR, -RC/-NR）

(a) -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 **オプションで指定できる処理種別を次に示します。処理種別の指定は省略できません。省略した場合，致命的エラー**（F0012）**となります。**

表 5-3 -R **で指定できる処理種別**

処理種別	内容
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。**
K [n] [M] (n = 1, 2, 4)	**スタティック・モデルにおいて，関数引数，および** auto **変数（**static auto **変数を除く）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。**
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。**
C	**構造体内の（**2 **バイト以上の）メンバを偶数番地に配置するためのアライン・データを挿入しません。つまり，構造体のパッキングを行います。**

備考　処理種別は，複数指定できます。

- -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。

表 5-4 -NR 指定時の解釈

処理種別	内容
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。
K	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	構造体メンバのパッキングを行いません。

【使用例】

```
C>cc78k0  -c054  -rds
```

(b) -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。**

表 5-5 **変数の最大幅（**-RD**）**

n **の値**	**割り付けられる変数の型**
1	char, unsigned char
2	char, unsigned char, short, unsigned short, int, unsigned int, enum, **データ・ポインタ**, **関数ポインタ（バンク機能（**-MF**）未使用時）**
4	char, unsigned char, short, unsigned short, int, unsigned int, enum, **全てのポインタ，** long, unsigned long
M	**構造体，共用体，配列**
省略	**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。**

(c) -RK/-NR

【記述形式】

-RK [n] [M]（n = 1, 2, 4 ） -NR

- **省略時解釈**

  -NR

【機能】

- -RK **オプションは，関数引数，および** auto **変数（**static auto **変数を除く）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，**-RK **オプションを無効にします。**

【用途】

- **スタティック・モデルにおいて，関数引数，および** auto **変数（**static auto **変数を除く）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。**

表 5-6 **割り付けられる最大幅（**-RK**）**

n **の値**	**割り付けられる変数の型**
1	char, unsigned char
2	char, unsigned char, short, unsigned short, int, unsigned int, enum, **データ・ポインタ**, **関数ポインタ（バンク機能（**-MF**）未使用時）**
4	char, unsigned char, short, unsigned short, int, unsigned int, enum, **全てのポインタ，** long, unsigned long
M	**構造体，共用体，配列**
省略	**すべての変数**

- register **宣言した変数には割り付けられません。**
- sreg **宣言された変数は，**-RK **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関数引数，および** auto **変数は，**sreg **宣言された関数引数，および** auto **変数と同様に扱います。**

【注意】

- **このオプションは，**-SM **オプションを指定した場合のみ有効です。**-SM **の指定がない場合，ワーニング・メッセージを出力し無視します。**

(d) -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。**

表 5-7 **割り付けられる最大幅（**-RS**）**

n **の値**	**割り付けられる変数の型**
1	char, unsigned char
2	char, unsigned char, short, unsigned short, int, unsigned int, enum, **データ・ポインタ** , **関数ポインタ（バンク機能（**-MF**）未使用時）**
4	char, unsigned char, short, unsigned short, int, unsigned int, enum, **全てのポインタ，**long, unsigned long
M	**構造体，共用体，配列**
省略	**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。**

(4) 最適化指定

最適化指定（-Q/-NQ）

【記述形式】

-Q [最適化種別]（複数の種別を指定する場合は続けて指定します） -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。詳細は表 5-8 を参照してください。

【説明】

- -Q オプションで指定できる最適化種別を表 5-8 に示します。

表 5-8 最適化種別

最適化種別	処理内容
省略	-QCJLVW が指定されたと見なします。
U	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よくします。
C [n] (n = 1, 2)	char に関する演算を汎整数拡張なしに行うことにより，コード効率がよくなります。 汎整数拡張とは，int 未満の整数（char/short）の演算は int に格上げして演算を行うように定めた ANSI-C の規定を指します[注]。 n の値によって，汎整数拡張しない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 ： 変数のみ汎整数拡張しない 2 ： 変数と定数を汎整数拡張しない
R [n] (n = 1, 2)	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。 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
J	分岐命令の最適化を行います。

表 5-8 最適化種別

最適化種別	処理内容
X [n] (n = 1-3)	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 ： ディフォールトとして，-Q を指定したものと見なす 3 ： コード・サイズ優先として，-QCJL4VW を指定したものと見なす
E	[HL + B] を使用したオブジェクトを出力します。 -SM オプション指定時のみ有効です。
H	[HL] .bit を使用したオブジェクトを出力します。
W [n] (n = 1, 2)	演算式の実行順序入れ替えなどを行うことでレジスタの有効活用を図り，効率の良いコードを出力します（例：2 項演算子の左辺式と右辺式の実行順序入れ替えなど）。したがって，（ANSI-C の仕様は，一部の演算子を除いて評価順序を定めないので，仕様の範囲内ですが）このオプションを付けた場合と付けない場合で，実行結果が異なる場合があります。ANSI-C の仕様に準じて，正しく作られたソースで問題となることはありません。 n の値によって，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 ： 演算式の実行順序入れ替えを行う 2 ： 1 に加えて，saddr 配置の char/unsigned char/short/unsigned short/int/unsigned int 配列を unsigned char 変数で参照する場合，配列のサイズを 256 バイト以下と仮定し，桁上がりを行わずアドレス計算を行う
V	オートマティック変数をレジスタ，saddr 領域へ自動的に割り当てます。
L [n] (n = 1-4)	定型コード・パターンをライブラリに置き換えます。 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 3 ： 2 に加えて，long 型のロード・ストア，DE/HL 間接参照コード 4 ： 3 に加えて 1 命令単位

注 CC78K0 では，-QC オプション指定時に，定数と文字定数の型を次のように取り扱っています。

0 〜 127 , 0x00 〜 0x7F , 00 〜 0177	char 型
128 〜 255 , 0x80 〜 0xFF , 0200 〜 0377	unsigned char 型
0U 〜 255U	unsigned char 型
‘ \0 ’ 〜 ‘ \377 ’	char 型

ただし，-QU オプション指定時には，‘ \200 ’ 〜 ‘ \377 ’ の範囲の文字定数は unsigned char 型の定数として扱い，+128 〜 +255 の値を持ちます。

また，-（マイナス）を付加した定数は，次のように扱います。

-0 〜 128	char 型
-129 〜	int 型

定数，および変数演算結果が，オーバフローする場合は定数，または変数のどちらかを演算結果が表現できる型にキャストしてください。キャストによりデータ型が変更されることを回避できます。-QC1 指定時は，定数演算に関しては符号拡張します。

- 最適化種別は複数指定できます。
- -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。
- ディフォールト・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を有効とします。
- -Q オプションが同時にいくつか指定された場合，最後に指定したものを有効とします。
- -QR と -SM の両オプションが指定された場合は，ワーニング・メッセージを出力し，-QR は無視されます。

【使用例】

- 修飾子なしの char を unsigned と見なすよう最適化を行います。

```
C>cc78k0  -c054  prime.c  -qu
```

- 次のように -QC と -QR の両オプションを指定すると，-QC オプションは無効になり，-QR オプションが有効となります。

```
C>cc78k0  -c054  prime.c  -qc  -qr
```

- -QC/-QR オプション両方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>cc78k0  -c054  prime.c  -qcr
```

(5) ディバグ情報出力指定

ディバグ情報出力指定（-G/-NG）

【記述形式】

```
-G [ n ]（ n = 1, 2 ）
-NQ
```

- 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・ファイル中にディバ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と，ディ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・ディバ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。

表 5-9 n の値による動作の違い

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・ファイル中のみディバグ情報（$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，ディバグ情報が付加されるため，ソース・ディバグも可能です。
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・ファイル中にディバ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を有効とします。
- 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】

- -G オプションを指定します。

```
C>cc78k0  -c054  prime.c  -g
```

(6) プリプロセス・リスト・ファイル作成指定

プリプロセス・リスト・ファイル作成指定（-P, -K）

(a) -P

【記述形式】

-P [出力ファイル名]

- 省略時解釈

  なし（ファイルを出力しない）

【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 plus 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 ” のコンボ・ボックスに出力先を指定してください。
- 個別オプション指定時は，出力ファイル名の変更もできます。
- ［出力］タブの “ 出力ファイル ” のコンボ・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>cc78k0  -c054  prime.c  -psample.ppl
```

(b) -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 **オプションで指定できる処理種別を次に示します。**

表 5-10 -K **オプションの処理種別**

処理種別	内容
省略	FLN **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を有効とします。**

【使用例】

- **プリプロセス・リスト** prime.ppl **のコメントの削除，行番号，およびページング処理を行います。**

```
C>cc78k0  -c054  prime.c  -p  -kcn
```

prime.ppl **を参照します。**

```
/*
78K/0 Series C Compiler VX.XX Preprocess List
                                                    Date : XX XXX XXXX Page :   1

Command         : -c054 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     : uPD78054
Device file     : VX.XX
*/
```

(7) プリプロセス指定

プリプロセス指定（-D, -U, -I）

(a) -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】

- 特定の定数をあるマクロ名にすべて置き換え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，一度に複数のマクロ定義を行えます。
- “ = ”と“ , ”の前後には空白を許しません。
- 定義名が省略されると，名前は“ 1 ”として定義されます。
- -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を有効とします。

【使用例】

```
C>cc78k0  -c054  prime.c  -dTEST , TIME = 10
```

(b) -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 **度に複数のマクロ定義を無効にできます。また，“ , ”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コンパイラの持つシステム・マクロ名は，**-U **オプションで無効にできません。**
- -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を有効とします。**

【使用例】

- -D, -U **オプションで同名のマクロを指定します。この例では，**TEST **マクロを無効にしています。**

```
C>cc78k0  -c054  prime.c  -dTEST  -uTEST
```

(c) -I

【記述形式】

-I ディレクトリ [, ディレクトリ] …（複数指定可能）

- 省略時解釈

  (i) ソース・ファイルのあるディレクトリ[注 1]

  (ii) 環境変数 INC78K0 により指定されたディレクトリ

  (iii) C:\NECTools32\inc78k0 [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ディレクトリから入力するよう指示します。

【用途】

- インクルード・ファイルをあるディレクトリ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，一度に複数のディレクトリを指定できます。
- “ , ” の前後には空白を入れることはできません。
- -I に続いてディレクトリ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。

  (i) ソース・ファイルのあるディレクトリ[注 1]

  (ii) -I オプションにより指定されたディレクトリ

  (iii) 環境変数 INC78K0 により指定されたディレクトリ

  (iv) C:\NECTools32\inc78K0 [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・ファイルのあるディレクトリ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:\NECTools32 にインストールした場合の例です。

【使用例】

- -I オプションを指定します。

```
C>cc78k0  -c054  prime.c  -ib: , b:\sample
```

(8) アセンブラ・ソース・モジュール・ファイル作成指定

アセンブラ・ソース・モジュール・ファイル作成指定（-A, -SA）

(a) -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。
- 出力ファイル
  .asm（：英数字）

【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 plus 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 ” のコンボ・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。
- 個別オプション指定時は，出力ファイル名の変更もできます。
- ［出力］タブの “ 出力ファイル ” のコンボ・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】

- アセンブラ・ソース・モジュール・ファイル sample.asm を作成します。

```
C>cc78k0  -c054  prime.c  -asample.asm
```

(b) -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。
- **出力ファイル**
  .asm（：英数字）

【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 plus **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 ” のコンボ・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。**
- **個別オプション指定時は，出力ファイル名の変更もできます。**
- **［出力］タブの “ 出力ファイル ” のコンボ・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】

- -SA **オプションを指定します。**

```
C>cc78k0  -c054  prime.c  -sa
```

prime.asm **を参照します。**

```
; 78K/0 Series C Compiler Vx.xx Assembler Source
;                                                    Date : xx xxx xxxx Time : xx : xx : xx

; Command                 : -c054 prime.c -sa
; In-file                 : prime.c
; Asm-file                : prime.asm
; Para-file               :

$PROCESSOR ( 054 )
$DEBUG
$NODEBUGA
$KANJICODE SJIS
$TOL_INF      03FH , 0330H , 02H , 020H , 00H

$DGS    FIL_NAM ,       .file ,         034H , 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          _mark
        PUBLIC          _main
        PUBLIC          _printf
        PUBLIC          _putchar
          :
@@CODE CSEG
_main :
$DGL   1 , 14
        push            hl                                      ; [ INF ] 1 , 4
        push            ax                                      ; [ INF ] 1 , 4
        push            ax                                      ; [ INF ] 1 , 4
        push            ax                                      ; [ INF ] 1 , 4
        push            ax                                      ; [ INF ] 1 , 4
        movw            ax , sp                                 ; [ INF ] 2 , 8
        movw            hl , ax                                 ; [ INF ] 1 , 4
??bf_main :
; line      9             : int i , prime , k , count ;
; line      10            :
; line      11            : count = 0 ;
$DGL  0 , 4
        mov             a , #00H        ; 0                     ; [ INF ] 2 , 4
        mov             [ hl ] , a ; count                      ; [ INF ] 1 , 4
```

```
        mov     [ hl + 1 ] , a ; count                  ; [ INF ] 2 , 8
; line  12      :
; line  13      : for ( i = 0 ; i <= SIZE ; i++ )
$DGL  0 , 6
        mov     [ hl + 6 ] , a      ; i                 ; [ INF ] 2 , 8
        mov     [ hl + 7 ] , a      ; i                 ; [ INF ] 2 , 8
?L0003 :
        mov     a , [ hl + 6 ]      ; i                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 7 ]      ; i                 ; [ INF ] 2 , 8
        cmpw    ax , #0C8H          ; 200               ; [ INF ] 3 , 6
        or1     CY , a.7                                ; [ INF ] 2 , 4
        bc      $$ + 4                                  ; [ INF ] 2 , 6
        bnz     $?L0004                                 ; [ INF ] 2 , 6
          :
        END

; *** Code Information ***
;
; $FILE H : \um\prime.c
;
; $FUNC main ( 8 )
;       bc = ( void )
;       CODE SIZE = 218 bytes , CLOCK_SIZE = 678 clocks , STACK_SIZE = 14 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 ( 31 )
;       bc = ( pointer s : ax , int i : [ sp + 2 ] )
;       CODE SIZE = 30 bytes , CLOCK_SIZE = 116 clocks , STACK_SIZE = 8 bytes
;
; $FUNC printf ( 41 )
;       bc = ( char c : x )
;       CODE SIZE = 14 bytes , CLOCK_SIZE = 58 clocks , STACK_SIZE = 6 bytes

; Target chip : uPD78054
; Device file : Vx.xx
```

コメントとして C ソースが付加されています。

(9) エラー・リスト・ファイル作成指定

エラー・リスト・ファイル作成指定（-E, -SE）

(a) -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。
- 出力ファイル
  .ecc（：英数字）

【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 plus 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 ” のコンボ・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。
- 個別オプション指定時は，出力ファイル名の変更もできます。
- ［出力］タブの “ 出力ファイル ” のコンボ・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】

- -E オプションを指定します。

```
C>cc78k0  -c054  prime.c  -e
```

エラー・リスト・ファイルを参照します。

```
prime.c ( 18 ) : CC78K0 warning W0745 : Expected function prototype
prime.c ( 20 ) : CC78K0 warning W0745 : Expected function prototype
prime.c ( 26 ) : CC78K0 warning W0622 : No return value
prime.c ( 37 ) : CC78K0 warning W0622 : No return value
prime.c ( 44 ) : CC78K0 warning W0622 : No return value

Target chip : uPD78054
Device file : Vx.xx

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

(b) -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。
- **出力ファイル**
  *.cer **：*.c ファイルに対するエラー・リスト（*：英数字）**
  *.her **：*.h ファイルに対するエラー・リスト**
  *.er **：*.c, *.h ファイル以外に対するエラー・リスト**

【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，ディレクトリ，およびファイル名の指定はできません。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 plus 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 ” のコンボ・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。**
- **個別オプション指定時は，出力ファイル名の変更もできます。**
- **［出力］タブの “ 出力ファイル ” のコンボ・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】

- -SE **オプションを指定します。**

```
C>cc78k0  -c054  prime.c  -se
```

prime.cer **を参照します。**

```
/*
78K/0 Series C Compiler VX.XX Error List                Date : XX XXX XXXX Time : XX : XX : XX

Command         : -c054  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 ) ;
*** CC78K0 warning W0745 : Expected function prototype
         count++ ;
         if ( ( count%8 ) == 0 ) putchar ( ' \n ' ) ;
*** CC78K0 warning W0745 : Expected function prototype
         for ( k = i + prime ; k <= SIZE ; k += prime )
            :
}
```

(10) クロスレファレンス・リスト・ファイル作成指定

クロスレファレンス・リスト・ファイル作成指定（-X）

【記述形式】

-X [出力ファイル名]

- 省略時解釈

  クロスレファレンス・リスト・ファイルを出力しません。

- 出力ファイル

  .xrf（：英数字）

【機能】

- -X オプションは，クロスレファレンス・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。クロスレファレンス・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】

- クロスレファレンス・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，クロスレファレンス・リスト・ファイル名は，“ 入力ファイル名 .xrf ” となります。
- 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，クロスレファレンス・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 plus 使用時に，出力先を変更する場合は，［出力］タブの “ クロス・レファレンス・リスト・ファイルの出力 [-x](S) ” の “ 出力パス ” のコンボ・ボックスに出力先を指定してください。
- 個別オプション指定時は，出力ファイル名の変更もできます。
- ［出力］タブの “ 出力ファイル ” のコンボ・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】

- -X オプションを指定します。

```
C>cc78k0  -c054  prime.c  -x
```

prime.xrf **を参照します。**

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

Command         : -c054 prime -x
In-file         : prime.c
Xref-file       : prime.xrf
Para-file       :

ATTRIB   MODIFY   TYPE   SYMBOL      DEFIN   REFERENCE

EXTERN          array     mark          5       14      16      22
EXTERN          func      main          7
REG1            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          func      printf        28      18      25
EXTERN          func      putchar       39      20
REG1            pointer   s             29      36
PARAM
REG1            int       i             30      35
PARAM
AUTO1           int       j             32      35
AUTO1           pointer   ss            33      36
REG1            char      c             40      43
PARAM
AUTO1           char      d             42      43
                #define   TRUE          1       14
                #define   FALSE         2       22
                #define   SIZE          35      13      15      21


 Target chip    : uPD78054
 Device file    : VX.XX
```

(11) リスト形式指定

リスト形式指定（-LW, -LL, -LT, -LF, -LI）

(a) -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 **オプションで指定できる文字数の範囲は，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】

- -LW **オプションを省略した場合のクロスレファレンス・リスト・ファイルを，ファイル名** .xrf **に出力します。**

```
C>cc78k0  -c054  prime.c  -x
```

(b) -LL

【記述形式】

-LL [行数]

- **省略時解釈**

  -LL66（**コンソール出力の場合は** 65535 **文字とします**）

【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 **オプションで指定できる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，**1 **ページの行数は** 66 **行となります（コンソール出力の場合は，**65535 **行とします）。**
- **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】

- **クロスレファレンス・リスト・ファイルの** 1 **ページの行数を** 20 **行と指定します。**

```
C>cc78k0  -c054  prime.c  -x  -ll20
```

(c) -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 **オプションで指定できる文字数の範囲は次のとおりです。**

  0 **≦指定できる文字数≦** 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 **オプションを省略します。**

```
C>cc78k0  -c054  prime.c  -p
```

- HT **コードによるブランクを** 1 **に指定します。**

```
C>cc78k0  -c054  prime.c  -p  -lt1
```

(d) -LF

【記述形式】

```
-LF
```

- **省略時解釈**

  なし

【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】

- -LF **オプションを指定します。**

```
C>cc78k0  -c054  prime.c  -a  -lf
```

(e) -LI

【記述形式】

```
-LI
```

- **省略時解釈**

  なし

【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】

- -LI **オプションを指定します。**

```
C>cc78k0  -c054  prime.c  -sa  -li
```

(12) ワーニング出力指定

ワーニング出力指定（-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ことを指示します。**

【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。または詳細なメッセージを出力させることもできます。**

【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。**

表 5-11 **ワーニング・メッセージのレベル**

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 **は，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）に出力しないことを指示します。**

【使用例】

- -W **オプションを省略した場合のワーニング・メッセージを参照します。**

```
C>cc78k0  -c054  prime.c
```

(13) 実行状態表示指定

実行状態表示指定（-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態をコンソールに出力しながら実行を行うときに指定します。**

【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です。**

【使用例】

- -V **オプションを指定します。**

```
C>cc78k0  -c054  prime.c  -v
```

(14) パラメータ・ファイル指定

パラメータ・ファイル指定（-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，コマンド行ではコンパイラ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・ファイル中に記述できる文字数の制限はありません。
- 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者優先です。
- “ ; ”，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  -c054  -aprime.asm  -e  -x
```

prime.pcc を使用してコンパイルします。

```
C>cc78k0  -fprime.pcc
```

(15) テンポラリ・ファイル作成ディレクトリ指定

テンポラリ・ファイル作成ディレクトリ指定（-T）

【記述形式】

-T ディレクトリ

- 省略時解釈
  環境変数 TMP により指定されたドライブ，ディレクトリに作成します。指定されていない場合は，カレント・ドライブ，カレント・ディレクトリに作成されます。

【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，ディレクトリを指示します。

【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ディレクトリ下にテンポラリ・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・ファイルをディレクトリ TMP に出力するよう指定します。

```
C>cc78k0  -c054  prime.c  -ttmp
```

(16) ヘルプ指定

ヘルプ指定（--/-?/-H）

【記述形式】

```
--/-?/-H
```

- 省略時解釈

  表示しない

【機能】

- --/-?/-H オプションは，オプションの簡単な説明，ディフォールト・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 plus 上では，このオプションは指定しないでください。PM plus 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。C コンパイラ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改行キーを，表示を途中で終了する場合には，改行キー以外の文字を入力したあとに改行キーを入力してください。

【使用例】

- -H オプションを指定します。

```
C>cc78k0  -h
```

(17) 機能拡張指定

機能拡張指定（-Z/-NZ）

【記述形式】

-Z [種別]（複数の種別を指定する場合は続けて指定します） -NZ

- 省略時解釈

  -NZ

【機能】

- -Z オプションは，種別指定に対応する処理を有効とします。
- -NZ オプションは，-Z オプションを無効とします。
- 種別は省略できません。省略した場合は，致命的エラー（F0012）となります。

【用途】

- 78K シリーズ拡張機能に対し，表 5-12 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。

表 5-12 -Z オプションの種別指定

種別指定	説明
省略時	-NZ が指定されたものとみなします。
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。

表 5-12 -Z オプションの種別指定

種別指定	説明
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 / callf / noauto / 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。 - 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。 （CC78K0 warning W0787: Bit field type is char） - -QC, -ZP, -ZC, -ZI, -ZL 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。 （CC78K0 warning W0029 : ‘ -QC ’ option is not portable） （CC78K0 warning W0031 : ‘ -ZP ’ option is not portable） （CC78K0 warning W0032 : ‘ -ZC ’ option is not portable） （CC78K0 warning W0036 : ‘ -ZI ’ option is not portable） （CC78K0 warning W0037 : ‘ -ZL ’ 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。 （CC78K0 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。 （CC78K0 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。 （CC78K0 error E0801 : Undefined control など）
M [n] （n = 1, 2）	スタティック・モデルの拡張仕様を使用可能にします。 引数の数を int サイズで 6 個，char サイズで 9 個まで記述可能とします。 1, 2 バイトの構造体／共用体引数，および関数返り値に，構造体／共用体返り値を記述可能とします。 n の値によって，_@KREGxx の使用方法を変更します。n を省略した場合は，n = 1 として解釈されます。 1 ：leaf 関数のみ共有領域として _@KREGxx を使用する。 2 ：_@KREGxx の退避／復帰を行って，_@KREGxx に引数，オートマティック変数を割り当てる。
D	関数の前後処理を，ライブラリに置き換えます。 -QL4 に対しワーニングを出力し，-QL3 として処理します。
R	自動的にパスカル関数修飾子を付加します。
F	フラッシュ用オブジェクトを出力します。
I	int，および short の記述は char とみなします。コンパイラ定義マクロ _FROM_INT_TO_CHAR_ は 1 とします。
L	long の記述は int とみなします。コンパイラ定義マクロ _FROM_LONG_TO_INT は 1 とします。

注 S, E, N は同時に指定できません。

【使用例】

- -ZC , -ZP オプションを指定します。

```
C>cc78k0  -c054  prime.c  -zpc
```

(18) デバイス・ファイルのサーチ・パス

デバイス・ファイルのサーチ・パス（-Y）

【記述形式】

-Y ディレクトリ

- 省略時解釈

  通常のサーチパスのみ

【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。
  通常のサーチ・パスは，次のとおりです。

  (i) < ..\dev >（cc78k0.exe の起動されたパスに対して）

  (ii) CC78K0 の起動されたパス

  (iii) カレント・ディレクトリ

  (iv) 環境変数 PATH

【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ディレクトリ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 plus 使用時は，〈プロジェクトの設定〉ダイアログの “ デバイス名 ” でデバイス・ファイルを登録する際にディレクトリが決定されます。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】

- -Y オプションを指定します。

```
C>cc78k0  -c054  -ya : \tmp\dev
```

(19) スタティック・モデル指定

スタティック・モデル指定（-SM）

【記述形式】

-SM [n]（ n = 1-16 ）

- **省略時解釈**

  ノーマル・モデル（n = 0）

【機能】

- **コンパイル時に** -SM **オプションを指定します。この際のオブジェクトをスタティック・モデルと呼び，**-SM **オプション無指定時のオブジェクトをノーマル・モデルと呼びます。**
- **通常，スタック・フレームをアクセスする命令よりも，静的領域をアクセスする命令の方が短く高速なので，オブジェクト・コードの短縮，実行速度の向上が図れます。**
- **ノーマル・モデルで行っている，**saddr **領域を使用している引数，および変数（割り込み関数でのレジスタ変数，**norec **関数の引数／オートマティック変数，ランタイム・ライブラリの引数）の退避／復帰処理を行わないので，割り込み処理の高速化が図れます。**
- **複数の** leaf **関数でデータを共有するので，メモリを節約できます。**

【用途】

- **オブジェクトの実行速度の向上，割り込み処理の高速化を図る場合に** -SM **オプションを指定し，オブジェクトをスタティック・モデルにします。**

【説明】

- **関数引数は，すべてレジスタで渡し，関数本体側では，関数引数，およびオートマティック変数を静的領域に割り付けます。**
- leaf **関数の場合，引数，およびオートマティック変数は，**FEDFH **以下の** saddr **領域に記述順に上位アドレスから割り付けます。この** saddr **領域は全モジュールの** leaf **関数で共有するため共有領域と呼びます。**
- n **の値が共有領域のサイズとなります。**
- n = 0 **のとき，および省略時は，共有領域を持ちません。**
- **コンパイラ定義マクロ** __STATIC_MODEL__ **は** 1 **とします。**
- **関数引数，およびオートマティック変数に，**sreg/__sreg **キーワードを付加できます。**sreg/__sreg **キーワードを付加した関数引数，およびオートマティック変数は** saddr **に割り付けられ，ビット操作が可能となります。**
- -RK **オプションを指定することにより，関数引数，およびオートマティック変数（関数内** static **変数を除く）を** saddr **に割り付け，ビット操作が可能となります。**

【注意】

- **引数／オートマティック変数を静的に確保しているので，再帰関数は引数／オートマティック変数の内容が破壊される可能性があります。直接自分自身を呼び出す場合はエラーとしますが，他の関数を呼び出した先で自分自身が呼び出された場合，コンパイラはそれを検出できず，エラーとなりません。**
- **割り込み時に処理中の関数が，割り込み処理（割り込み関数，および割り込み関数が呼び出す関数）により呼び出された場合，引数／オートマティック変数の内容が破壊される可能性があります。**
- **割り込み時に処理中の関数が共有領域を使用している場合でも，共有領域の退避／復帰は行いません。**

【使用例】

```
C>cc78k0  -c054  test.c  -sm16
```

(20) 関数情報ファイル指定

関数情報ファイル指定（-MF）

【記述形式】

-MF ファイル名

- 省略時解釈

  全てのソースを共通領域に配置

- 出力ファイル

  .fin（：英数字）

【機能】

- -MF オプションは，関数情報ファイルの参照，および生成を指示します。

【用途】

- 関数のバンク領域，または共通領域への配置を指示します。

【説明】

- 関数情報ファイルの編集方法については，「CC78K0 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。

【注意】

- リンク対象の全ての C ソース・ファイルに対し，同一の関数情報ファイルを指定してください。

【使用例】

- funcinf.fin を使用してコンパイルします。

```
C>cc78k0  -cf053664  -mffuncinf.fin
```

第 6 章　C コンパイラの出力ファイル

この章では，CC78K0 が出力するファイルについて説明します。

CC78K0 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
- クロスレファレンス・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，ディバグ情報出力指定オプション（-G）によりディバ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】

```
        ; 78K/0 Series C Compiler V （1） x.xx Assembler Source
        ;                                                    Date : （2） xxxxx Time : （3） xxxxx

        ; Command          : （4） -c054 prime.c –sa
        ; In-file          : （5） prime.c
        ; Asm-file         : （6） prime.asm
        ; Para-file        : （7）

        $PROCESSOR ( （8） 054 )
（9）    $DEBUG
（10）   $NODEBUGA
（11）   $KANJICODE SJIS
（12）   $TOL_INF 03FH , 0330H , 02H , 020H , 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 , 4
（14）             push  ax                                    ; （21） [ INF ] 1 , 4
（14）             push  ax                                    ; （21） [ INF ] 1 , 4
（14）             push  ax                                    ; （21） [ INF ] 1 , 4
                     :
（18） ??bf_main :
                     :
        ; （22）  *** Code Information ***
        ;
        ; （23）  $FILE C:\NECTools32\Smp78k0\CC78K0\prime.c
        ; （24）  $FUNC main ( 8 )
        ; （25）  bc = ( void )
        ; （26）  CODE SIZE = 218 bytes , CLOCK_SIZE = 678 clocks , STACK_SIZE = 14 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）$FUNC printf ( 31 )
;（25）bc = ( pointer s :ax , int i : [ sp + 2 ] )
;（26）CODE SIZE = 30 bytes , CLOCK_SIZE = 116 clocks , STACK_SIZE = 8 bytes
;
;（24）$FUNC putchar ( 41 )
;（25）bc = ( char c : x )
;（26）CODE SIZE = 14 bytes , CLOCK_SIZE = 58 clocks , STACK_SIZE = 6 bytes

; Target chip :（19）uPD78054
; Device file ：（20）Vx.xx
```

【出力項目の説明】

表 6-1 出力項目の説明（アセンブラ・ソース・モジュール・ファイル）

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 “ CC78K0 ” 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。 デバイス・ファイルに関する資料を参照してください。
(9)	ディバグ情報	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)	アセンブラのディバグ情報の制御	9（固定）	NODEBUGA コントロールを出力します。 $NODEBUGA を出力します。

表 6-1 出力項目の説明（アセンブラ・ソース・モジュール・ファイル）

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（$DGS で始まる情報）。 ディバグ情報出力指定オプションを指定した場合にのみ出力します。ただし，-G1 が指定された場合は出力しません。
(14)	アセンブラ・ソース本体	−	コンパイル結果のアセンブラ・ソースを出力します。
(15)	行番号	4（固定）	C ソース・モジュール・ファイルの行番号（右詰めゼロ・サプレスの 10 進数）です。
(16)	C ソース	−	入力 C ソース・イメージです。80 カラム目からは，次行の 16 カラム目から出力します。ただし，1 カラム目に “ ; ” を出力します。
(17)	ライン・ナンバ情報	−	ライン・ナンバ・エントリ用の行番号（$DGL で始まる情報）です。 ディバ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レーベル	最大 34（可変）	関数のレーベル情報です（?? で始まる情報）。 ディバ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）。
(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。
(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。
(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）。

表 6-1 出力項目の説明（アセンブラ・ソース・モジュール・ファイル）

項番	内容	桁数	形式
(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】

```
/*
78K/0 Series C Compiler V（1）x.xx Error List                    Date :（2）xxxxx Time :（3）xxxxx

Command         :（4）-c054 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 ;
 *** CC78K0 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 ) ;
 *** CC78K0 warning（9）W0745 :（10）Expected function prototype
            :
/*
  （11）Target chip : uPD78054
  （12）Device file : Vx.xx
Compilation complete ,     （13）1 error ( s ) and     （14）5 warning ( s ) found.
*/
```

【出力項目の説明】

表 6-2 出力項目の説明（C ソース付きのエラー・リスト・ファイル）

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 “ CC78K0 ” 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。 # は内部エラーの場合は C，致命的エラーの場合は F，文法の誤りやコンパイラの制限によるエラーの場合は E，ワーニングの場合は 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 ) : CC78K0 warning（3）W0745 :（4）Expected function prototype
（1）prime.c (（2）20 ) : CC78K0 warning（3）W0745 :（4）Expected function prototype
（1）prime.c (（2）26 ) : CC78K0 warning（3）W0622 :（4）No return value
（1）prime.c (（2）37 ) : CC78K0 warning（3）W0622 :（4）No return value
（1）prime.c (（2）44 ) : CC78K0 warning（3）W0622 :（4）No return value

Target chip :（7）uPD78054
Device file :（8）Vx.xx

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

【出力項目の説明】

表 6-3 出力項目の説明（エラー・メッセージのみのエラー・リスト・ファイル）

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。 # は内部エラーの場合は C，致命的エラーの場合は F，文法の誤りやコンパイラの制限によるエラーの場合は E，ワーニングの場合は 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＞

```
/*
78K/0 Series C Compiler V（1）x.xx Preprocess List     Date :（2）xxxxx Page :（3）xxx

Command :（4）-c054  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 : uPD78054
  （11）Device file : Vx.xx
*/
```

【出力項目の説明】

表 6-4 出力項目の説明（プリプロセス・リスト・ファイル）

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“ CC78K0 ”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。

表 6-4 出力項目の説明（プリプロセス・リスト・ファイル）

項番	内容	桁数	形式
(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。
(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 クロスレファレンス・リスト・ファイル

クロスレファレンス・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＞

```
78K/0 Series C Compiler V（1）x.xx Cross reference List Date :（2）xxxxx Page :（3）xxx

Command       :（4）-c054  prime.c -x -lw80
In-file       :（5）prime.c
Xref-file     :（6）prime.xrf
Para-file     :（7）
Inc-file      : [ n ] （8）

ATTRIB        MODIFY   TYPE  SYMBOL    DEFINE  REFERENCE

（9）EXTERN    （10）     （11）array   （12）mark    （13）5  （14）14 （14）16 （14）22
（9）EXTERN    （10）     （11）func    （12）main    （13）7
（9）AUTO1     （10）     （11）int     （12）i       （13）9  （14）13 （14）13 （14）13  （14）14
                                                    （14）15 （14）15 （14）15 （14）16
                                                    （14）17 （14）17 （14）21
（9）AUTO1     （10）     （11）int     （12）prime   （13）9  （14）17 （14）18 （14）21  （14）21
（9）AUTO1     （10）     （11）int     （12）k       （13）9  （14）21 （14）21 （14）21  （14）22
（9）AUTO1     （10）     （11）int     （12）count   （13）9  （14）11 （14）19 （14）20  （14）25
  :
/*（15）Target chip : uPD78054
  （16）Device file : Vx.xx  */
```

【出力項目の説明】

表 6-5 出力項目の説明（クロスレファレンス・リスト・ファイル）

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 “ CC78K0 ” 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)	クロスレファレンス・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。
(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，レーベルは 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），CALLF（callf 関数），NOAUTO（noauto 関数），NOREC（norec 関数），SREG（sreg•bit 変数），RWSFR（sfr 変数），ROSFR（リード・オンリの sfr 変数），WOSFR（ライト・オンリーの sfr 変数），VECT（割り込み関数），BANK（バンク関数）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。

表 6-5 出力項目の説明（クロスレファレンス・リスト・ファイル）

項番	内容	桁数	形式
(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。
(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，CC78K0 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）

CC78K0 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。

- 正常終了時　　：0
- WARNING あり　　：0
- FATAL ERROR あり　　：1
- ABORT 時　　：2

PM plus を利用せず，コマンド行で CC78K0 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られます。

【使用例】

```
cc78k0 -c054 %1
IF ERRORLEVEL 1 GOTO ERR
cc78k0 -c054 %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できます。

7.2 開発環境を整える（環境変数）

CC78K0 は，次の環境変数をサポートしています。

- PATH ：実行形式のサーチ・パス
- INC78K0 ：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（-ZE, -ZS, -ZN オプションでも指定可能）
  （euc：EUC コード，sjis：シフト JIS コード，none：2 バイト・コードなし）
- LIB78K0 ：ライブラリのサーチ・パス

【使用例】

＜ DOS プロンプト使用時の場合＞

```
; AUTOEXEC.BAT
PATH C:\NECTools32\bin ; C:\bat ; C:\cc78k0 ; C:\tool
VERIFY ON
BREAK ON
SET INC78K0 = C:\NECTools32\inc78k0
SET LIB78K0 = C:\NECTools32\lib78k0
SET TMP = C:\tmp
SET LANG78K = sjis
```

【説明】

- パス指定により，C:\NECTools32\bin , C:\bat , C:\cc78k0 , 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:\NECTools32\inc78k0 から検索されます。
  設定しない場合，C:\NECTools32\inc78k0（C:\NECTools32 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:\NECTools32\lib78k0 から検索されます。
  設定しない場合，C:\NECTools32\lib78K0（C:\NECTools32 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 plus の利用時に，環境変数の設定はしないでください。

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しに，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 plus 上でビルド（MAKE）を中止したい場合は，PM plus ウインドウ上の［ビルド］メニュー内の［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 plus 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

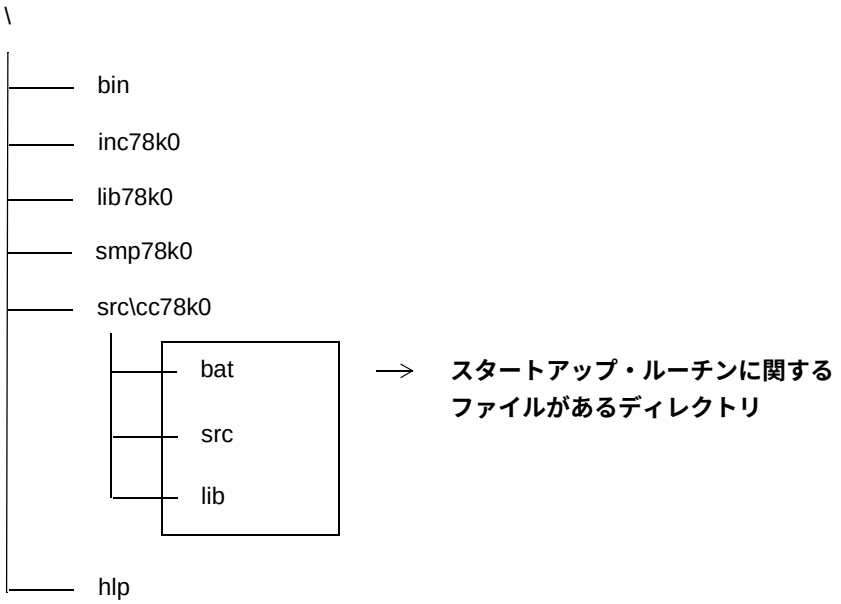
C 言語によるプログラムを実行させるには，システムへ組み込むための ROM 化処理，ユーザ・プログラム（main 関数）の起動などを行うプログラムが必要となります。このプログラムのことをスタートアップ・ルーチンと呼びます。

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。CC78K0 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，コンパイラ・パッケージのディレクトリ src\cc78k0 に格納されています。



次に src\cc78k0 以下にあるディレクトリの内容について示します。

lib ディレクトリには，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，78K0 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。CC78K0 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。
ファイル内容については，「2.6.4 ライブラリ・ファイル」を参照してください。

8.1.1 ディレクトリ bat の内容

このディレクトリのバッチ・ファイルは，PM plus 上では使用できません。

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ご使用ください。

なお，bat ディレクトリ下のデバイス・ファイル（d002.78k と d014.78k）は，ライブラリ他更新用バッチ・ファイル起動時に使用するもので，開発用ではありません。開発時にはデバイス・ファイルが必要です。

表 8-1 ディレクトリ “ 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，ない（レジスタの退避のみ行う）setjmp/longjmp を作成します。

8.1.2 ディレクトリ src の内容

ディレクトリ 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 ディレクトリのバッチ・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・ファイルを作成できます。

表 8-2 ディレクトリ “ 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。

8.2 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，ディレクトリ bat にある mkstup.bat を使用します。

また，mkstup.bat では，RA78K0　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 のあるディレクトリ src\cc78k0\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 「デバイス・ファイルに関する資料」を参照してください。

【使用例】

- 対象品種が uPD78054 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  054
```

バッチ・ファイル mkstup.bat は，次のようにディレクトリ bat と同じ階層のディレクトリ 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。
それぞれのディレクトリには，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 ——— s0.rel
          s0b.rel
          s0e.rel
          s0l.rel
          s0lb.rel
          s0le.rel
          s0sm.rel
          s0smb.rel
          s0sme.rel
          s0sml.rel
          s0smlb.rel
          s0smle.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。よって，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図 8-1 に示します。

図 8-1 スタートアップ・ルーチンの概要

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，表 8-3 に示します。

表 8-3 スタートアップ・ルーチン・ソース間の違い

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

CC78K0 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，表 8-4 に示します。

表 8-4 ソース・ファイルとオブジェクト・ファイルの対応

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	s0*.rel 注 2, 3
ROM 化ファイル	rom.asm	ライブラリに含まれます。

注 1　*：標準ライブラリ未使用の場合は n が付加されます，使用の場合は文字は付加されません。

注 2　“ b ”はブート領域用，“ e ”はフラッシュ領域用スタートアップ・ルーチンです。

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。

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レー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 (1) 〜 (6)（後記参照）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)
                                                                        ; (3)
BRKSW       EQU     1       ; brk , sbrk , calloc , free , malloc , realloc function use
EXITSW      EQU     1       ; exit , atexitfunction use
$_IF ( _STATIC )
RANDSW      EQU     0       ; rand , srandfunction use
DIVSW       EQU     0       ; divfunction use
LDIVSW      EQU     0       ; 1divfunction use
FLOATSW     EQU     0       ; floating point variables use
$ELSE
RANDSW      EQU     1       ; rand , srandfunction use
DIVSW       EQU     1       ; divfunction use
LDIVSW      EQU     1       ; 1divfunction use
FLOATSW     EQU     1       ; floating point variables use
$ENDIF
STRTOKSW    EQU     1       ; strtokfunction use

PUBLIC      _@cstart , _@cend

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

$_IF ( EXITSW )
```

```
_@FNCTBL :    DS      2 * 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。ディフォールトは，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。よって，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は，「**RA78K0 **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）。**

【リンク時に -sSTACK **を指定した場合の例】**

lk78k0.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 **はディフォールトで配置されますので，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，smp78k0\CC78K0 **ディレクトリの** lk78k0.dr **を参考にしてください。**

```
                    先頭アドレス  サイズ
                          ↓       ↓
memory  SDR         : ( 0FE20h , 00098h )
memory  STACK       : ( xxxxh , xxxh )   ←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 lk78k0.dr を指定します（例：-dlk78k0.dr）
merge  @@INIS       : = SDR
merge  @@DATS       : = SDR
merge  @@BITS       : = SDR
```

(5) ROM 化処理用レーベルの外部参照宣言

ROM 化処理用レーベルは，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 の初期設定にある (1) ～ (4) について説明します。

【cstart.asm の初期設定】

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

@LCODE      CSEG
_@cstart :
            SEL    RB0                                              ; (2)
                                                                    ; (3)
            MOVW   SP , #_@STBEG          ; SP <-stack begin address
            CALL   !_hdwinit                                        ; (4)
$ENDIF
              :
$_IF ( BRKSW OR EXITSW OR RANDSW OR FLOATSW )
            MOVW   AX , #0
$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) **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。**

(3) 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。**

(4) **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】

```
; ************************************
; ROM DATA COPY
; ************************************
; copy external variables having initial value
        MOVW   HL , #_@R_INIT
        MOVW   DE , #_@INIT
LINIT1 :
        MOVW   AX , HL
        CMPW   AX , #_?R_INIT
        BZ     $LINIT2
        MOV    A , [ HL ]
        MOV    [ DE ] , A
        INCW   HL
        INCW   DE
        BR     $LINIT1
LINIT2:
        MOVW   HL , #_@DATA
; copy external variables which doesn't have initial value
LDATA1 :
            :
```

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。

- 図 8-2 に（a）初期値あり外部変数の ROM 化処理を示します。
  変数（a）の初期値は，コンパイラにより，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 セグメントの先頭レーベル，最終レーベルは，_@R_INIT , _?R_INIT で，@@INIT セグメントの先頭レーベル，最終レーベルは，_@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（表 8-6 を参照）。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，最終レーベルを表 8-5 と表 8-6 に示します。

図 8-2 ROM 化処理

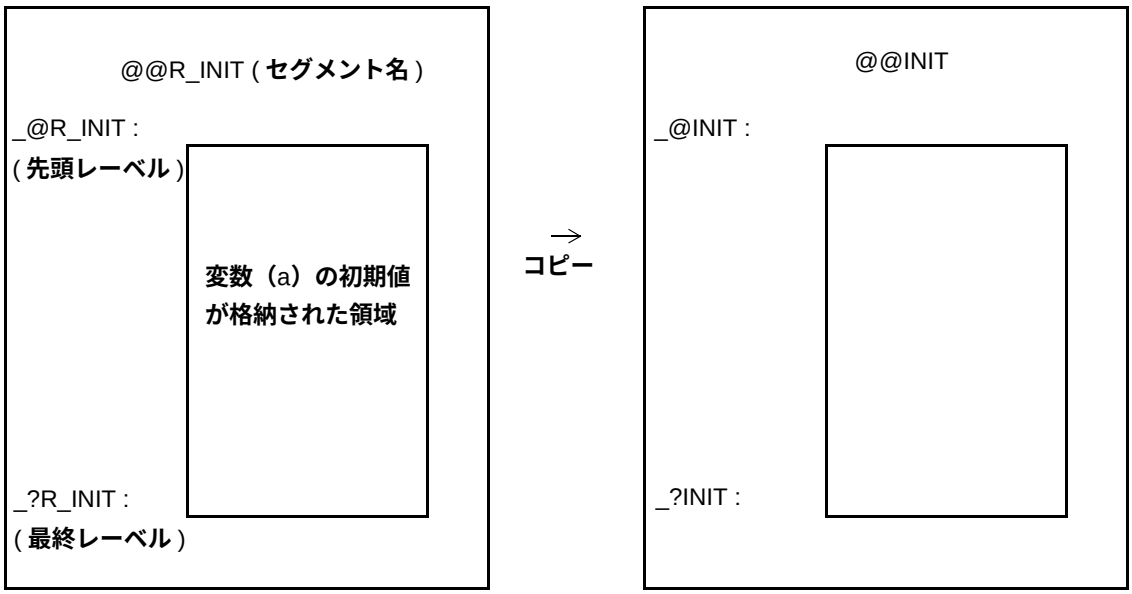


表 8-5 初期値の ROM 領域

変数の種類	セグメント	先頭レーベル	最終レーベル
初期値あり外部変数（a）	@@R_INIT	_@R_INIT	_?R_INIT
初期値あり sreg 変数（c）	@@R_INIS	_@R_INIS	_?R_INIS

表 8-6 初期値の RAM 領域（コピー先）

変数の種類	セグメント	先頭レーベル	最終レーベル
初期値あり外部変数（a）	@@INIT	_@INIT	_?INIT
初期値なし外部変数（b）	@@DATA	_@DATA	_?DATA
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 )
        MOVW    AX , #0
        CALL    !_exit          ; exit ( 0 ) ;          ; (2)
$ENDIF
        BR      $$
;
_@cend :
                                                        ; (3)
@@R_INIT        CSEG    UNITP
_@R_INIT :
@@R_INIS        CSEG    UNITP
_@R_INIS :
@@INIT          DSEG    UNITP
_@INIT :
@@DATA          DSEG    UNITP
_@DATA :
@@INIS          DSEG    SADDRP
_@INIS :
@@DATS          DSEG    SADDRP
_@DATS :
@@CALT          CSEG    CALLT0
@@CALF          CSEG    FIXED
@@CNST          CSEG    UNITP
@@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，レーベルの定義

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，レーベルを定義します。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。レーベルは，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 , _?R_INIS
        PUBLIC _?INIT , _?DATA , _?INIS , _?DATS
;
@@R_INIT        CSEG    UNITP                                   ; (1)
_?R_INIT :
@@R_INIS        CSEG    UNITP
_?R_INIS :
@@INIT          DSEG    UNITP
_?INIT :
@@DATA          DSEG    UNITP
_?DATA :
@@INIS          DSEG    SADDRP
_?INIS :
@@DATS          DSEG    SADDRP
_?DATS :
;
        END
```

(1) ROM 化処理で使用するレーベルの定義

ROM 化処理で，（a）～（d）の変数（「(3) ROM 化処理」を参照）ごとに，使用するレーベルを定義します。これらのレーベルは，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

CC78K0 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，ディレクトリ src\cc78k0\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

表 8-7 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 は削除できません（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できます。

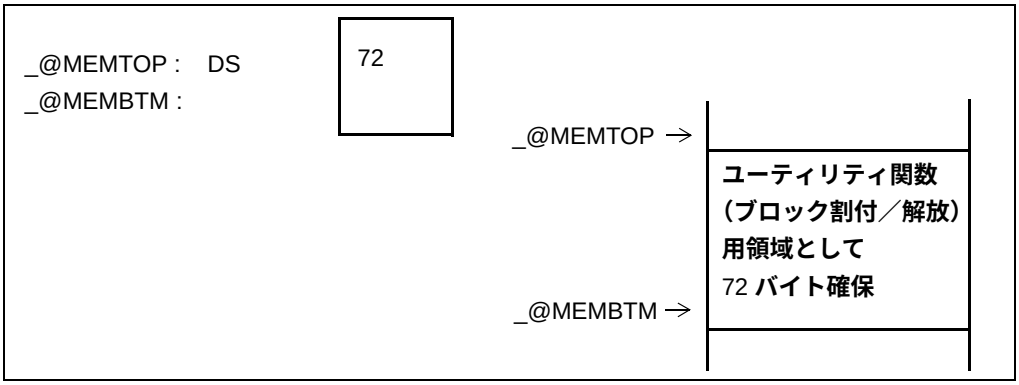
表 8-7 ライブラリ関数で使われるシンボル

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。

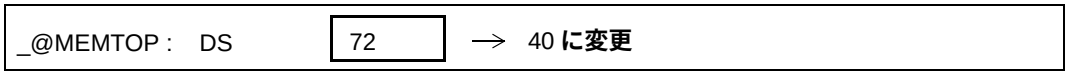
例　ユーティリティ関数（ブロック割付／解放）用として，72 バイト確保したい場合，スタートアップ・ルーチンの初期設定を，次のように修正してください。



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，「**RA78K0 **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）。**

- CC78K0 **は，ショート・ダイレクト・アドレス領域（**saddr **領域）の一部を，次のようなコンパイラ固有の目的で使用する場合があります。具体的には，ノーマル・モデルの場合は，**FEB8H-FEDFH **までの** 40 **バイトの領域です。**-SM [n] **オプションにより，スタティック・モデルを指定した場合は，共有領域として** saddr **領域の一部** [FED0H-FEDFH] **を使用します。**

  （ノーマル・モデル）

  (a) **ランタイム・ライブラリの引数** [FEB8H-FEBFH]

  (b) norec **関数の引数，自動変数** [FEC0H-FECFH]

  (c) -qr **オプションを指定した場合の** register **変数** [FED0H-FEDFH]

  (d) **標準ライブラリの作業用 （（**b**）と（**c**）の領域の一部）**

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

  （スタティック・モデル）

  (a) **共有領域** [FED0H-FEDFH]

次にリンク・ディレクティブ・ファイル（lk78k0.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。

＜ lk78k0.dr ＞

先頭アドレス サイズ	
memory RAM : (0FB00h , 00320h)	→ このサイズを大きくします。
memory SDR : (0FE20h , 00098h)	（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できます（詳しくは，「CC78K0 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らなくなった場合は，対応する memory 文を変更してください。

(3) RTOS を使用する場合

RX78K0，および CC78K0 は，それぞれに初期化処理ルーチンをサンプルとして提供しています（アセンブラ形式）。したがって，RX78K0 と CC78K0 を併用する場合には，それぞれに必要な処理を 1 つの初期化処理ルーチン内に含めるように変更しなければなりません。
ここでは，その修正方法の例を，cstart.asm（CC78K0 提供初期化処理ルーチン）に対して，startup.asm（RX78K0 提供初期化ルーチン）内に記述してある処理を追加していく形で説明します。なお，CC78K0 は Ver.3.50 以降を想定しています。

備考　cstart.asm は，ROM 化処理なし，標準ライブラリ使用版となっています。

(i) RX78K0 で必要な次の EXTRN 宣言を追加します。

＜変更後＞

```
EXTRN  sys_inf , ?sysrt
```

(ii) cstart.asm に記述してある，main 関数，および exit 関数の EXTRN 宣言を削除します。スタック領域をユーザが確保する場合（イニシャル・タスク以外のタスクのスタックを使用する場合）は，_@STBEG の EXTRN 宣言も削除します（_@STBEG 領域は，リンク時に -s オプションを指定することにより自動確保されます）。

＜変更前＞

```
        EXTRN       _main , _@STBEG , _hdwinit
$_IF ( EXITSW )
        EXTRN       _exit
$ENDIF
```

＜変更後＞

```
EXTRN       _@STBEG , _hdwinit
```

また，EXITSW **の設定箇所の変更も行います。**

＜変更前＞

```
EXITSW      EQU   1
```

＜変更後＞

```
EXITSW      EQU   0
```

(iii) RX78K0 **で提供している** vcttbl.asm **のベクタ** 0 **と重複しないように，次の箇所の修正（または** vcttbl.asm **の修正）を行います。**_@cstart **を使用しない場合は，使用するシンボル（アドレス）に変更してください。**

＜変更前＞

```
@@VECT00    CSEG   AT  0
            DW      _@cstart
```

(iv) **レジスタバンクの選択を行う前に，割り込み禁止状態にします。**

＜変更前＞

```
SEL     RB0
```

＜変更後＞

```
DI
SEL     RB0
```

(v) **スタック領域の** _@STBEG **を使用しない場合は，次の箇所を変更します。**

＜変更前＞

```
MOVW   SP , #_@STBEG            ; SP <- stack begin address
```

(vi) **ハードウエア初期化関数（**hdwinit**）に，ユーザ・システムで必要となるハードウエアの初期化処理を記述してください。**

(vii) RX78K0 **を使用する場合，**main **関数，および** exit **関数は必要ないため，次の箇所を削除します。また，**RX78K0 **が制御する上で不必要となる処理も削除し，**RX78K0 **のシステム初期化用ルーチンに制御を移す処理を追加します。**

＜変更前＞

```
            CALL    !_ma   in           ; main ( ) ;
$_IF ( EXITSW )
            MOVW   AX , #0
            CALL    !_exit              ; exit ( 0 ) ;
$ENDIF
            BR      $$
```

＜変更後＞

```
MOVW   HL , #sys_inf
MOV    A , [ HL ]
MOV    X , A
MOV    A , [ HL + 1 ]
BR     AX
```

＜修正後の初期化処理ルーチンの例＞

```
; Copyright ( C ) NEC Electronics Corporation 19xx , 20xx
; NEC ELECTRONICS CONFIDENTIAL AND PROPRIETARY
; All rights reserved by NEC Electronics Corporation.
; This program must be used solely for the purpose for which
; it was furnished by NEC Electronics Corporation. No part of this
; program may be reproduced or disclosed to others , in any
; form , without the prior written permission of NEC Electronics
; Corporation. Use of copyright notice does not evidence
; publication of the program.
;=====================================
;       W-1     cstart
;
;       Version x.xxxx    Xxx 20xx
;=====================================
          NAME    @cstart

$INCLUDE ( def.inc )
$INCLUDE ( macro.inc )

; ---------------------------------------------------------------------------------------------------------------------
; declaration of symbol
;
; attention ) :         change to EQU value 1 -> 0  if necessary
; ---------------------------------------------------------------------------------------------------------------------
;
BRKSW           EQU   1           ; brk , sbrk , calloc , free , malloc , realloc function use
EXITSW          EQU   0           ; exit , atexit   function use   ; 変更個所
$_IF ( _STATIC )
RANDSW          EQU   0           ; rand , srand    function use
DIVSW           EQU   0           ; div             function use
LDIVSW          EQU   0           ; ldiv            function use
FLOATSW         EQU   0           ; floating point variables use
$ELSE
RANDSW          EQU   1           ; rand , srand  function use
DIVSW           EQU   1           ; div             function use
LDIVSW          EQU   1           ; ldiv            function use
FLOATSW         EQU   1           ; floating point variables use
$ENDIF
STRTOKSW        EQU   1           ; strtok          function use
        PUBLIC              _@cstart , _@cend
```

```
$_IF ( BRKSW )
        PUBLIC          _@BRKADR , _@MEMTOP , _@MEMBTM
$ENDIF
$_IF ( EXITSW )
        PUBLIC          _@FNCTBL , _@FNCENT
$ENDIF
$_IF ( RANDSW )
        PUBLIC          _@SEED
$ENDIF
$_IF ( DIVSW )
        PUBLIC          _@DIVR
$ENDIF
$_IF ( LDIVSW )
        PUBLIC          _@LDIVR
$ENDIF
$_IF ( STRTOKSW )
        PUBLIC          _@TOKPTR
$ENDIF
$_IF ( BRKSW OR FLOATSW )
        PUBLIC          _errno
$ENDIF

; ------------------------------------------------------------------------------------------------------------------------
; external declaration of symbol for stack area
;
; _@STBEG has value of the end address +1 of compiler's stack area.
; _@STBEG is created by linker with -S option.
; Accordingly , specify the -S option when linking.
; ------------------------------------------------------------------------------------------------------------------------
        EXTRN           sys_inf , ?sysrt                ; 追加箇所
        EXTRN           _@STBEG , _hdwinit              ; 変更箇所

; ------------------------------------------------------------------------------------------------------------------------
; external declaration of label for ROMable
; ------------------------------------------------------------------------------------------------------------------------
        EXTRN           _?R_INIT , _?R_INIS , _?DATA , _?DATS

; ------------------------------------------------------------------------------------------------------------------------
; allocation area which library uses
;
; _@FNCTBL      top address of area used in atexit function
; _@FNCENT      total number of functions registered by the atexit function
; _@SEED        sequence of pseudo-random numbers
; _@DIVR        a result of div library
; _@LDIVR       a result of ldiv library
; _@TOKPTR      pointer which strtok function uses
```

```
; _errno               errno number code
; _@MEMTOP             top address of area which memory management functions use
; _@MEMBTM             bottom address of area which memory management functions use
; _@BRKADR             break value
; ------------------------------------------------------------------------------------------------------------------------
@@DATA                 DSEG    UNITP

$_IF ( EXITSW )
_@FNCTBL :             DS      2 * 32
_@FNCENT :             DS      2
$ENDIF
$_IF ( RANDSW )
_@SEED ;               DS      4
$ENDIF
$_IF ( DIVSW )
_@DIVR :               DS      4
$ENDIF
$_IF ( LDIVSW )

_@LDIVR :              DS      8
$ENDIF
$_IF ( STRTOKSW )
_@TOKPTR :             DS      2
$ENDIF
$_IF ( BRKSW OR FLOATSW )
_errno :               DS      2
$ENDIF
$_IF ( BRKSW )
_@BRKADR :             DS      2
_@MEMTOP :             DS      32
_@MEMBTM :
$ENDIF

@@VECT00               CSEG    AT    0                    ; 必要あれば変更
        DW             _@cstart                           ;

@LCODE                 CSEG

_@cstart:
; ------------------------------------------------------------------------------------------------------------------------
; setting the register bank RB0 as work register set
; ------------------------------------------------------------------------------------------------------------------------
        DI                                                ; 追加箇所
        SEL     RB0
; ------------------------------------------------------------------------------------------------------------------------
; setting the stack pointer
;
```

```
; _@STBEG is created by linker with -S option.
; ------------------------------------------------------------------------------------------------------------------------
        MOVW            sp , #_@STBEG ; SP <- stack begin address    ; 必要あれば変更
        CALL            !_hdwinit
; ------------------------------------------------------------------------------------------------------------------------
; errno and _@FNCENT are initialized to 0
; The positive error number will be set by several libraries at called them.
; ------------------------------------------------------------------------------------------------------------------------
$_IF ( BRKSW OR EXITSW OR RANDSW OR FLOATSW )
        MOVW            AX , #0
$ENDIF
$_IF ( BRKSW OR FLOATSW )
        MOVW            _errno , AX             ; errno <- 0
$ENDIF
$_IF ( EXITSW )
        MOVW            !_@FNCENT , AX     ; FNCENT <- 0
$ENDIF
; ------------------------------------------------------------------------------------------------------------------------
; initializing _@SEED as 1
;
; Pseudo-random sequence is decided by _@SEED value. This value can be set by
; srand function. If rand is called before srand , the random sequence will
; be the same as when srand is called with a _@SEED value as 1 at first.
; ------------------------------------------------------------------------------------------------------------------------
$_IF ( RANDSW )
        MOVW            !_@SEED + 2 , AX
        INC             X
        MOVW            !_@SEED , AX        ; SEED <- 1
$ENDIF
; ------------------------------------------------------------------------------------------------------------------------
; setting _@MEMTOP address to _@BRKADR
; ------------------------------------------------------------------------------------------------------------------------
$_IF ( BRKSW )
        MOVW            AX , #_@MEMTOP
        MOVW            !_@BRKADR , AX     ; BRKADR <- #MEMTOP
$ENDIF

; ------------------------------------------------------------------------------------------------------------------------
;  ROM data copy
; ------------------------------------------------------------------------------------------------------------------------

; copy external variables having initial value
        MOVW            HL , #_@R_INIT
        MOVW            DE , #_@INIT
LINIT1 :
        MOVW            AX , HL
        CMPW            AX , #_?R_INIT
```

```
        BZ          $LINIT2
        MOV         A , [ HL ]
        MOV         [ DE ] , A
        INCW        HL
        INCW        DE
        BR          $LINIT1
LINIT2 :
        MOVW        HL , #_@DATA
; copy external variables which doesn't have initial value
LDATA1 :
        MOVW        AX , HL
        CMPW        AX , #_?DATA
        BZ          $LDATA2
        MOV         A , #0
        MOV         [ HL ] , A
        INCW        HL
        BR          $LDATA1
LDATA2 :
; copy sreg variables having initial value
        MOVW        HL , #_@R_INIS
        MOVW        DE , #_@INIS
LINIS1 :
        MOVW        AX , HL
        CMPW        AX , #_?R_INIS
        BZ          $LINIS2
        MOV         A , [ HL ]
        MOV         [ DE ] , A
        INCW        HL
        INCW        DE
        BR          $LINIS1
LINIS2 :
        MOVW        HL , #_@DATS
; copy sreg variables which doesn't have initial value
LDATS1 :
        MOVW        AX , HL
        CMPW        AX , #_?DATS
        BZ          $LDATS2
        MOV         A , #0
        MOV         [ HL ] , A
        INCW        HL
        BR          $LDATS1
LDATS2 :
; ------------------------------------------------------------------------------------------------------------------------
; branches to the reset routine for system initialization of RX78K0
; ------------------------------------------------------------------------------------------------------------------------
        MOVW        HL , #sys_inf                                   ;
```

```
        MOV             A , [ HL ]                                  ;
        MOV             X , A                                       ; 変更箇所
        MOW             A , [ HL ]                                  ;
        BR              AX                                          ;
;
_@cend :
; ------------------------------------------------------------------------------------------------------------------------
; define segment and label used by ROMable processing
; ------------------------------------------------------------------------------------------------------------------------
@@R_INIT        CSEG    UNITP
_@R_INIT :
@@R_INIS        CSEG    UNITP
_@R_INIS :
@@INIT          DSEG    UNITP
_@INIT :
@@DATA          DSEG    UNITP
_@DATA :
@@INIS          DSEG    SADDRP
_@INIS :
@@DATS          DSEG    SADDRP
_@DATS :
@@CALT          CSEG    CALLT0
@@CALF          CSEG    FIXED
@@CNST          CSEG    UNITP
@@BITS          BSEG
;
END
```

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 8-8 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	先頭レーベル	終端レーベル
初期値あり外部変数（a）	@ER_INIT CSEG UNITP	E@R_INIT	E?R_INIT
初期値あり sreg 変数（c）	@ER_INIS CSEG UNITP	E@R_INIS	E?R_INIS

表 8-9 コピー先の RAM 領域のセクション

変数の種類	セグメント	先頭レーベル	終端レーベル
初期値あり外部変数（a）	@EINIT DSEG UNITP	E@INIT	E?INIT
初期値なし外部変数（b）	@EDATA DSEG UNITP	E@DATA	E?DATA
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レーベルを付けます。

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

- 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レーベルを付けます。

  E?R_INIT, E?R_INIS, E?INIT, E?DATA, E?INIS, E?DATS

- 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レーベルのアドレスから，終端レーベルのアドレス -1 までの内容を，RAM 領域の各セグメントの先頭レー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。

第 9 章　エラー・メッセージ

この章では，CC78K0 **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)　　: CC78K0 error E0712 : Declaration syntax
prime.c (8)　　: CC78K0 error E0301 : Syntax error
prime.c (8)　　: CC78K0 error E0701 : External definition syntax
prime.c (19)　　: CC78K0 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] CC78K0 error C0101: Internal error
[xxx.c < yyy > zzz] CC78K0 error C0103 : Intermediate file error
[xxx.c < yyy > zzz] CC78K0 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

コンパイラが出力するエラー・メッセージには，次の 10 種類があります。

(1) コマンド行に対するエラー・メッセージ
(2) 内部エラー，メモリに対するエラー・メッセージ
(3) 文字に対するエラー・メッセージ
(4) 構成要素に対するエラー・メッセージ
(5) 変換に対するエラー・メッセージ
(6) 式に対するエラー・メッセージ
(7) 文に対するエラー・メッセージ
(8) 宣言，関数定義に対するエラー・メッセージ
(9) 前処理指令に対するエラー・メッセージ
(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

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コンパイラ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。

C/F/E/Wnnnn

C ：内部エラー

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。

F ：致命的エラー

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。

E ：文法の誤りやコンパイラの制限によるエラー

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。

W ：警告

コンパイルを続行します。

nnnn（4 桁の数字）

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。
	処理	Please enter ‘cc78k0 --’ 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。
	処理	Please enter ‘cc78k0 --’ 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・ファイルとしてすでに存在しているため，作成できません。
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，またはディレクトリが含まれています。
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。
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。
	処理	Please enter ‘cc78k0 --’ 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。

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	
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。
	処理	Please enter ‘cc78k0 --’ 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。
	処理	Please enter ‘cc78k0 --’ 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。
	処理	Please enter ‘cc78k0 --’ 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。
	処理	Please enter ‘cc78k0 --’ 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。
	処理	パラメータ・ファイルの中に，パラメータ・ファイルを指定できませんので，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。
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。
	コンパイラ	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。

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	
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。
	コンパイラ	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。
	処理	Please enter 'cc78k0 --' 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。
W0036	メッセージ	'-ZI' option is not portable
	原因	-ZI オプションは，ANSI 準拠ではありません（-ZI 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）。
W0037	メッセージ	'-ZL' option is not portable
	原因	-ZL オプションは，ANSI 準拠ではありません（-ZL 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）。
W0038	メッセージ	'-ZI' option specified - regarded as '-QC'
	原因	int と short を char とみなすオプション（-ZI）が指定されたため，int 拡張抑制最適化オプション（-QC）を有効とします。
W0039	メッセージ	'-SM' option specified - regarded as '-ZL'
	原因	スタティック・モデル指定オプション（-SM）が指定されたため，long を int とみなすオプション（-ZL）を有効とします。
W0040	メッセージ	'-RK' option required '-SM' - ignored '-RK'
	原因	ローカル変数最適化オプション（-RK）は，スタティック・モデル指定オプション（-SM）が指定されたときのみ有効です。ローカル変数最適化オプション（-RK）は無視されます。
W0041	メッセージ	'-SM' option specified - ignored '-QR'
	原因	スタティック・モデル指定オプション（-SM）が指定されたため，レジスタ最適化オプション（-QR）は無視されます。

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	
W0045	メッセージ	'-SM' option specified - ignored '-ZR'
	原因	スタティック・モデル指定オプション（-SM）が指定されたため，パスカル関数インタフェース指定オプション（-ZR）は無視されます。
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。
W0052	メッセージ	'-ZD' option specified -regarded as '-QL3'
	原因	プロローグ／エピローグ対応ライブラリ使用オプション（-ZD）が指定されたため，定型コード・パターンのライブラリ置換オプション（-QL）で，-QL4 は -QL3 とみなします。
W0054	メッセージ	'-ZF' option specified - ignored '-ZD'
	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，プロローグ／エピローグ対応ライブラリ使用オプション（-ZD）は無視されます。
W0055	メッセージ	'-ZM' option required '-SM' - ignored '-ZM'
	原因	スタティック・モデル拡張仕様オプション（-ZM）は，スタティック・モデル指定オプション（-SM）が指定されたときのみ有効です。-ZM オプションは無視されます。
W0056	メッセージ	This chip does not support bank function - ignored bank function
	原因	指定したデバイスはバンク機能をサポートしていません。 バンク機能は無視されます。
W0057	メッセージ	'-MF' option specified for bank function - ignored '-ZR'
	原因	バンク機能をサポートするための関数情報ファイル指定オプション（-MF）が指定されたため，パスカル関数インタフェース指定オプション（-ZR）は無視されます。
W0058	メッセージ	'-MF' option specified for bank function - ignored '-SM'
	原因	バンク機能をサポートするための関数情報ファイル指定オプション（-MF）が指定されたため，スタティック・モデル指定オプション (-SM) は無視されます。
W0059	メッセージ	'-MF' option specified for bank function - ignored '-ZM'
	原因	バンク機能をサポートするための関数情報ファイル指定オプション（-MF）が指定されたため，スタティック・モデル拡張仕様オプション (-ZM) は無視されます。
W0060	メッセージ	Function Information File : Illegal description for ' シンボル '
	原因	関数情報ファイル中の記述に不整合があります。
F0061	メッセージ	Function Information File : Syntax Error near ' エラー発生箇所 '
	原因	関数情報ファイルに構文エラーがあります。

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	
E0062	メッセージ	Function Information File : Unknown Mapping for ‘ 配置属性指定対象 ’
	原因	関数情報ファイル中に不正なマッピング属性が指定されています。
	処理	マッピング属性は，C，またはバンク番号を指定してください。
W0063	メッセージ	Function Information File : Function (‘ 関数名 ’) does not exist in ‘ ソース・ファイル名 ’
	原因	関数情報ファイル中にソース・ファイルには存在しない関数が指定されています。この関数情報指定は無視されます。
W0064	メッセージ	Function Information File : Deleted function (‘ 関数名 ’) in ‘ ソース・ファイル名 ’
	原因	ソース・ファイルには存在しない関数の記述を関数情報ファイルから削除しました。
W0065	メッセージ	‘-QW3’ option deleted - regarded as ‘-QW2’
	原因	最適化指定オプション (-QW3) は削除されたため，-QW2 を有効とします。
W0066	メッセージ	‘-QW5’ option deleted - regarded as ‘-QW4’
	原因	最適化指定オプション (-QW5) は削除されたため，-QW4 を有効とします。

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。
	処理	問い合わせ先にお問い合わせください。
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。
	コンパイラ	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。
	処理	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。
	処理	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。
	処理	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。
	処理	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。
	処理	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。

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	
E0113	メッセージ	Compiler limit：too many local lables
	原因	1 関数内の内部レーベル数が処理限界数を越えました。
	処理	関数本体が大き過ぎます。関数を分割してください。
E0115	メッセージ	Compiler limit : too much code defined in file for a bank
	原因	ファイル内のコードに割り当てられた領域がバンクのサイズの制限を越えました。
	処理	ファイル内のコード・サイズがバンクのサイズを越えないように，ファイル，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。
	処理	-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。
	処理	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。
	処理	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。
	処理	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。

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	
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	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。
W0330	メッセージ	‘-ZI’ option specified-int & short are treated as char in this file
	原因	-ZI オプションが指定されました。ファイル中の int，および short の記述は char とみなします。

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	
W0331	メッセージ	‘-ZL’ option specified-long is treated as int in this file
	原因	-ZL オプションが指定されました。ファイル中の long の記述は int とみなします。
W0333	メッセージ	‘-SM’ option specified - ignored ‘ 関数属性子 ’ keyword in this file
	原因	スタティック・モデル指定オプション（-SM）が指定されました。ファイル中の関数属性子は無視されます。
E0334	メッセージ	‘-SM’ option specified - float & double keywords are not allowed
	原因	スタティック・モデル指定オプション（-SM）が指定されました。float 型，および double 型は許されません。
W0335	メッセージ	‘-SM’ option specified - long constant is treated as int constant
	原因	スタティック・モデル指定オプション（-SM）が指定されました。long 定数の記述は int 定数とみなされます。
W0339	メッセージ	‘__temp’ required ‘-SM –ZM’ - ignored ‘ ‘__temp’ in this file
	原因	テンポラリ変数指定キーワード __temp は，スタティック・モデル指定オプション（-SM）と，スタティック・モデル拡張仕様オプション（-ZM）が指定されたときのみ有効です。 このファイルでは，__temp キーワードは無視されます。
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。
E0341	メッセージ	‘-MF’ option specified for bank function - ‘ 関数修飾子 ’ keyword is not allowed
	原因	バンク機能をサポートするための関数情報ファイル指定オプション (-MF) が指定されたため，この関数修飾子は使用できません。
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，ご注意ください。
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。

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 関数中では行えません。
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。
	処理	“ -> ” 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。

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	
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。
	処理	“ . ” 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られません。
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られません。
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。

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	
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。
	処理	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 noauto or norec function prototype
	原因	noauto, 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 noauto or norec function prototype
	原因	noauto, 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。

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	
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。
E0549	メッセージ	‘-SM’ option specified-recursive function is not allowed
	原因	スタティック・モデル指定オプション（-SM）が指定されました。再帰呼び出しは許されません。
E0550	メッセージ	Cannot call function in __flashf function
	原因	__flashf 関数の中からは関数を呼び出せません。
E0551	メッセージ	Cannot call long type library in __flashf function
	原因	__flashf 関数の中からは long 型のライブラリを呼び出せません。
W0552	メッセージ	Undeclared parameter in __flashf function prototype
	原因	__flashf 関数のプロトタイプ宣言において，パラメータが宣言されていません。

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 2048 を越えました。
E0603	メッセージ	Compiler limit：too many labels
	原因	レーベル数が 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 ‘ レーベル名 ’
	原因	switch 文の中で同じ case レーベルが 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 レーベルが 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。

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	
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。
	処理	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，出力ファイルが作成できません。

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 個まで宣言できます。
	処理	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 noauto or norec function
	原因	noauto, norec 関数のパラメータが多すぎます。
	処理	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。
W0708	メッセージ	Already declared symbol ‘ 変数名 ’
	原因	同じ変数がすでに宣言されています。
E0709	メッセージ	Different bank direction specified same file
	原因	同一ファイルに対して異なるバンク指定が行われました。
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。
	処理	シンボルの定義は 1 回にしてください。

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	
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。
	処理	レジスタ変数を減らしてください。使用可能な数については，「CC78K0 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。
E0716	メッセージ	Not allowed automatic data in noauto function
	原因	noauto 関数中ではオートマティック変数は使用できません。
E0717	メッセージ	Too many automatic data in noauto or norec function
	原因	noauto, norec 関数のオートマティック変数が多すぎます。
	処理	noauto, norec 関数のオートマティック変数を減らしてください。使用可能な数については，「CC78K0 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。
	処理	bit, boolean, __boolean 型変数を減らしてください。使用可能な数については，「CC78K0 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。
	コンパイラ	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 noauto or norec function
	原因	noauto, 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。

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	
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。
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。
	処理	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。

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	
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。
	処理	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。
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	関数は初期化できません。
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 宣言できません。
	コンパイラ	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。

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	
W0763	メッセージ	Cannot declare const with bit, boolean
	原因	bit, boolean 型変数は，const 宣言できません。
	コンパイラ	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。
	コンパイラ	extern 宣言を無視します。
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 noauto or norec function
	原因	noauto, norec 関数のオートマティック変数の型が大きいです。
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/callf/noauto/norec/__banked/__pascal are not allowed for interrupt function
	原因	interrupt 関数は，callt, callf, noauto, norec, __banked, __pascal 宣言は指定できません。
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は使用できません。
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/ret_wup except in rtos_interrupt_handler
	原因	RTOS_INTERRUPT ハンドラ以外で，ret_int/ret_wup システム・コールは呼び出せません。
E0777	メッセージ	Not call ret_int/ret_wup in rtos_interrupt_handler
	原因	RTOS_INTERRUPT ハンドラにおいて，ret_int/ret_wup システム・コールを呼び出していません。
E0778	メッセージ	Cannot call ext_tsk in interrupt function
	原因	割り込み関数／割り込みハンドラで，ext_tsk システム・コールは呼び出せません。
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。

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	
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。
E0781	メッセージ	‘-SM’ option specified-variable parameters are not allowed
	原因	スタティック・モデル指定オプション（-SM）が指定されました。可変長引数は許されません。
E0782	メッセージ	‘-SM’ option specified-structure & union parameter is not allowed
	原因	スタティック・モデル指定オプション（-SM）が指定されました。構造体，および共用体型の引数は許されません。
E0783	メッセージ	‘-SM’ option specified-structure & union return vallue is not allowed
	原因	スタティック・モデル指定オプション（-SM）が指定されました。構造体，および共用体型の返却値は許されません。
E0784	メッセージ	‘-SM’ option specified-too many parameters of function
	原因	スタティック・モデル指定オプション（-SM）が指定されました。関数引数が，3 引数 6 バイトの制限を越えています。
E0785	メッセージ	‘-SM’ option specified-expected function prototype
	原因	スタティック・モデル指定オプション（-SM）が指定されました。関数プロトタイプ宣言がありません。
W0786	メッセージ	‘-SM’ option specified-undeclared parameter in function prototype
	原因	スタティック・モデル指定オプション（-SM）が指定されました。関数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。
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。
E0788	メッセージ	Cannot allocate a __flash function ‘ 関数名 ’
	原因	__flash 関数は配置できません。
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できません。
E0790	メッセージ	Callt/callf/__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/callf/__interrupt 宣言は指定できません。
E0791	メッセージ	‘-ZF’ option specified - cannot allocate a callt/callf 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/callf 関数は配置できません。
W0792	メッセージ	Undeclared parameter in__pascal function definition or prototype
	原因	__pascal 関数定義，またはプロトタイプ宣言において，パラメータが宣言されていません。パラメータがない場合は void を明記する必要があります。

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	
W0793	メッセージ	Variable parameters are not allowed for__pascal function - ignored __pascal
	原因	__pascal 関数には可変長パラメータを指定できません。__pascal キーワードは無視されます。
E0794	メッセージ	Too many parameters in __flashf function
	原因	__flashf 関数のパラメータが多すぎます。
E0795	メッセージ	Illegal type for parameter in __flashf function
	原因	__flashf 関数のパラメータに許されない型を指定しています。
E0796	メッセージ	Too many automatic data in __flashf function
	原因	__flashf 関数のオートマティック変数が多すぎます。
E0797	メッセージ	Illegal type for automatic data in __flashf function
	原因	__flashf 関数のオートマティック変数に許されない型を指定しています。
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。
	処理	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ご確認ください。
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。
	処理	環境変数 INC78K0 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。

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	
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。
	コンパイラ	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。
	処理	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。

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	
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。
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。
	処理	#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 コンパイラ・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 or LEAFWORK 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。
	コンパイラ	レジスタ・バンク指定，スタック切り替え指定，あるいは LEAFWORK 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。
	コンパイラ	あと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。

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	
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。
	コンパイラ	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。
	コンパイラ	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 0x80-0xff80
	原因	フラッシュ領域分岐テーブルの先頭アドレスは 0x80-0xff80 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。

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	
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。
	コンパイラ	-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。
	処理	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	ファイルが open できません。
	処理	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できます。「5.4 デバイス・ファイルのサーチ・パス（-Y）」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・ファイルが open 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・ファイルが open 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。
	処理	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・ファイルが作成できません。
F0908	メッセージ	No available directory space
	原因	ドライブのディレクトリ・エリアの不足によりテンポラリ・ファイルが作成できません。
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・ファイルが作成できません。
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。 1. 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。 2. 内部矛盾により出力テンポラリ・ファイルを read only 属性で open しています。

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	
F0911	メッセージ	Reading unwritten data, no available directory space
	原因	次の理由により入出力エラーが発生しました。 1. EOF を越えて入力を行おうとしました。 2. ドライブのディレクトリ・エリアの不足によりテンポラリ・ファイルを作成できません。
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。
	処理	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。問い合わせ先へご連絡ください。
F0913	メッセージ	Requires MS-DOS V2.11 or greater
	原因	OS が MS-DOS（V2.11 以上）ではありません。
F0914	メッセージ	Insufficient memory in hostmachine
	原因	メモリ不足のためコンパイラを起動できません。
	処理	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しません。W0837 の処理を行ってください。
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 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。

付録 A　サンプル・プログラム

この章では，CC78K0 **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>cc78K0 -c054 prime.c -a –p -x -e -ng
```

```
78K/0 Series C Compiler Vx.xx [ xx xxx xxxx ]
Copyright ( C ) NEC Electronics Corporation xxxx , xxxx

sample\prime.c ( 18 ) : CC78K0 warning W0745 : Expected function prototype
sample\prime.c ( 20 ) : CC78K0 warning W0745 : Expected function prototype
sample\prime.c ( 26 ) : CC78K0 warning W0622 : No return value
sample\prime.c ( 37 ) : CC78K0 warning W0622 : No return value
sample\prime.c ( 44 ) : CC78K0 warning W0622 : No return value

Target chip : uPD78054
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

```
;78K/0 Series C Compiler Vx.xx Assembler Source
;                                                    Date : xx xxx xxxx Time : xx : xx : xx
; Command       : -c054 prime.c -a -p -x -e -ng
; In-file       : prime.c
; Asm-file      : prime.asm
; Para-file     :

$PROCESSOR  ( 054 )
$NODEBUG
$NODEBUGA
$KANJICODE SJIS
$TOL_INF        03FH , 0330H , 02H , 020H , 00H

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

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

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

; line    5
; line    8

@@CODE CSEG
_main :
        push    hl                                  ; [ INF ] 1 , 4
        push    ax                                  ; [ INF ] 1 , 4
        push    ax                                  ; [ INF ] 1 , 4
        push    ax                                  ; [ INF ] 1 , 4
        push    ax                                  ; [ INF ] 1 , 4
        movw    ax , sp                             ; [ INF ] 2 , 8
```

```
        movw    hl , ax                                 ; [ INF ] 1 , 4
; line  11
        mov     a , #00H        ; 0                     ; [ INF ] 2 , 4
        mov     [ hl ] , a      ; count                 ; [ INF ] 1 , 4
        mov     [ hl + 1 ] , a  ; count                 ; [ INF ] 2 , 8
; line  13
        mov     [ hl + 6 ] , a  ; i                     ; [ INF ] 2 , 8
        mov     [ hl + 7 ] , a  ; i                     ; [ INF ] 2 , 8
L0003 :
        mov     a , [ hl + 6 ]  ; i                     ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 7 ]  ; i                     ; [ INF ] 2 , 8
        cmpw    ax , #0C8H      ; 200                   ; [ INF ] 3 , 6
        orl     CY , a.7                                ; [ INF ] 2 , 4
        bc      $$ + 4                                  ; [ INF ] 2 , 6
        bnz     $L0004                                  ; [ INF ] 2 , 6
;line   14
        addw    ax , #_mark                             ; [ INF ] 3 , 6
        movw    de , ax                                 ; [ INF ] 1 , 4
        mov     a , #01H        ; 1                     ; [ INF ] 2 , 4
        mov     [ de ] , a                              ; [ INF ] 1 , 4
        mov     a , [ hl + 6 ]  ; i                     ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 7 ] , a ; i                  ; [ INF ] 2 , 8
        incw    ax                                      ; [ INF ] 1 , 4
        mov     [ hl + 7 ] , a  ; i                     ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     [ hl + 6 ] , a  ; i                     ; [ INF ] 2 , 8
        br      $L0003                                  ; [ INF ] 2 , 6
L0004 :
; line  15
        mov     a , #00H        ; 0                     ; [ INF ] 2 , 4
        mov     [ hl + 6 ] , a  ; i                     ; [ INF ] 2 , 8
        mov     [ hl + 7 ] , a  ; i                     ; [ INF ] 2 , 8
L0006 :
        mov     a , [ hl + 6 ]  ; i                     ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 7 ]  ; i                     ; [ INF ] 2 , 8
        cmpw    ax , #0C8H      ; 200                   ; [ INF ] 3 , 6
        or1     CY , a.7                                ; [ INF ] 2 , 4
        bc      $$ + 7                                  ; [ INF ] 2 , 6
        bz      $$ + 5                                  ; [ INF ] 2 , 6
        br      !L0007                                  ; [ INF ] 3 , 6
; line  16
        addw    ax , #_mark                             ; [ INF ] 3 , 6
```

```
        movw    de , ax                                 ; [ INF ] 1 , 4
        mov     a , [ de ]                              ; [ INF ] 1 , 4
        cmp     a , #00H        ; 0                     ; [ INF ] 2 , 4
        bz      $L0015v                                 ; [ INF ] 2 , 6
; line  17
        mov     a , [ hl + 6 ]  ; i                     ; [ INF ] 2 , 8
        rolc    a , 1                                   ; [ INF ] 1 , 2
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 7 ]  ; i                     ; [ INF ] 2 , 8
        rolc    a , 1                                   ; [ INF ] 1 , 2
        addw    ax , #03H       ; 3                     ; [ INF ] 3 , 6
        mov     [ hl + 5 ] , a  ; prime                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     [ hl + 4 ] , a  ; prime                 ; [ INF ] 2 , 8
; line  18
        xch     a , x                                   ; [ INF ] 1 , 2
        push    ax                                      ; [ INF ] 1 , 4
        movw    ax , #L0011                             ; [ INF ] 3 , 6
        call    !_printf                                ; [ INF ] 3 , 7
        pop     ax                                      ; [ INF ] 1 , 4
; line  19
        mov     a , [ hl ]      ; count                 ; [ INF ] 1 , 4
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 1 ]  ; count                 ; [ INF ] 2 , 8
        incw    ax                                      ; [ INF ] 1 , 4
        mov     [ hl + 1 ] , a  ; count                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     [ hl ] , a      ; count                 ; [ INF ] 1 , 4
; line  20
        xch     a , x                                   ; [ INF ] 1 , 2
        movw    _@RTARG0 , ax                           ; [ INF ] 2 , 6
        movw    ax , #08H       ; 8                     ; [ INF ] 3 , 6
        call    !@@isrem                                ; [ INF ] 3 , 7
        or      a , x                                   ; [ INF ] 2 , 4
        bnz     $L0012                                  ; [ INF ] 2 , 6
        movw    ax , #0AH       ; 10                    ; [ INF ] 3 , 6
        call    !_putchar                               ; [ INF ] 3 , 7
L0012 :
; line  21
        mov     a , [ hl + 4 ]  ; prime                 ; [ INF ] 2 , 8
        add     a , [ hl + 6 ]  ; i                     ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 5 ]  ; prime                 ; [ INF ] 2 , 8
        addc    a , [ hl + 7 ]  ; i                     ; [ INF ] 2 , 8
        mov     [ hl + 3 ] , a  ; k                     ; [ INF ] 2 , 8
```

```
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     [ hl + 2 ] , a      ; k                 ; [ INF ] 2 , 8
L0014 :
        mov     a , [ hl + 2 ]      ; k                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 3 ]      ; k                 ; [ INF ] 2 , 8
        cmpw    ax , #0C8H          ; 200               ; [ INF ] 3 , 6
        or1     CY , a.7                                ; [ INF ] 2 , 4
        bc      $$ + 4                                  ; [ INF ] 2 , 6
        bnz     $L0015                                  ; [ INF ] 2 , 6
; line   22
        addw    ax , #_mark                             ; [ INF ] 3 , 6
        movw    de , ax                                 ; [ INF ] 1 , 4
        mov     a , #00H            ; 0                 ; [ INF ] 2 , 4
        mov     [ de ] , a                              ; [ INF ] 1 , 4
        mov     a , [ hl + 4 ]      ; prime             ; [ INF ] 2 , 8
        add     a , [ hl + 2 ]      ; k                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 5 ]      ; prime             ; [ INF ] 2 , 8
        addc    a , [ hl + 3 ]      ; k                 ; [ INF ] 2 , 8
        mov     [ hl + 3 ] , a      ; k                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     [ hl + 2 ] , a      ; k                 ; [ INF ] 2 , 8
        br      $L0014                                  ; [ INF ] 2 , 6
L0015 :
; line   24
        mov     a , [ hl + 6 ]      ; i                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 7 ]      ; i                 ; [ INF ] 2 , 8
        incw    ax                                      ; [ INF ] 1 , 4
        mov     [ hl + 7 ] , a      ; i                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     [ hl + 6 ] , a      ; i                 ; [ INF ] 2 , 8
        br      !L0006                                  ; [ INF ] 3 , 6
L0007 :
; line   25
        mov     a , [ hl ]          ; count             ; [ INF ] 1 , 4
        xch     a, x                                    ; [ INF ] 1 , 2
        mov     a , [ hl + 1 ]      ; count             ; [ INF ] 2 , 8
        push    ax                                      ; [ INF ] 1 , 4
        movw    ax , #L0017                             ; [ INF ] 3 , 6
        call    !_printf                                ; [ INF ] 3 , 7
        pop     ax                                      ; [ INF ] 1 , 4
; line   26
        pop     ax                                      ; [ INF ] 1 , 4
```

```
        pop     ax                                      ; [ INF ] 1 , 4
        pop     ax                                      ; [ INF ] 1 , 4
        pop     ax                                      ; [ INF ] 1 , 4
        pop     hl                                      ; [ INF ] 1 , 4
        ret                                             ; [ INF] 1 , 6
; line  31
_printf :
        push    hl                                      ; [ INF ] 1 , 4
        push    ax                                      ; [ INF ] 1 , 4
        push    ax                                      ; [ INF ] 1 , 4
        push    ax                                      ; [ INF ] 1 , 4
        movw    ax , sp                                 ; [ INF ] 2 , 8
        movw    hl , ax                                 ; [ INF ] 1 , 4
; line  35
        mov     a , [ hl + 10 ]     ; i                 ; [ INF ] 2 , 8
        mov     [ hl + 2 ] , a      ; j                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 11 ]     ; i                 ; [ INF ] 2 , 8
        mov     [ hl + 3 ] , a      ; j                 ; [ INF ] 2 , 8
; line  36
        mov     a , [ hl + 4 ]      ; s                 ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     a , [ hl + 5 ]      ; s                 ; [ INF ] 2 , 8
        mov     [ hl + 1 ] , a      ; ss                ; [ INF ] 2 , 8
        xch     a , x                                   ; [ INF ] 1 , 2
        mov     [ hl ] , a          ; ss                ; [ INF ] 1 , 4
; line  37
        pop     ax                                      ; [ INF ] 1 , 4
        pop     ax                                      ; [ INF ] 1 , 4
        pop     ax                                      ; [ INF ] 1 , 4
        pop     hl                                      ; [ INF ] 1 , 4
        ret                                             ; [ INF ] 1 , 6
; line  41
_putchar :
        push    hl                                      ; [ INF ] 1 , 4
        push    ax                                      ; [ INF ] 1 , 4
        push    ax                                      ; [ INF ] 1 , 4
        movw    ax , sp                                 ; [ INF ] 2 , 8
        movw    hl , ax                                 ; [ INF ] 1 , 4
; line  43
        mov     a , [ hl + 2 ]      ; c                 ; [ INF ] 2 , 8
        mov     [ hl + 1 ] , a      ; d                 ; [ INF ] 2 , 8
; line  44
        pop     ax                                      ; [ INF ] 1 , 4
        pop     ax                                      ; [ INF ] 1 , 4
```

```
        pop     hl                                                      ; [ INF ] 1 , 4
        ret                                                             ; [ INF ] 1 , 6
        END


; ***Code Information***
;
; $FILE C:\NECTools32\sample\prime.c
;
; $FUNC main ( 8 )
;       bc = ( void )
;       CODE SIZE = 218 bytes , CLOCK_SIZE = 678 clocks , STACK_SIZE = 14 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 ( 31 )
;       bc = ( pointer s : ax , int i : [ sp + 2 ] )
;       CODE SIZE = 30 bytes , CLOCK_SIZE = 116 clocks , STACK_SIZE = 8 bytes
;
; $FUNC putchar ( 41 )
;       bc = ( char c : x )
;       CODE SIZE = 14 bytes , CLOCK_SIZE = 58 clocks , STACK_SIZE = 6 bytes

; Target chip : uPD78054
; Device file : Vx.xx
```

A.3.2 プリプロセス・リスト・ファイル

```
/*
78K/0 Series C Compiler Vx.xx Preprocess List                    Date : xx xxx xxxx Page : 1
Command         : -c054 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 : 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 ( s , i )
        29 : char           *s ;
        30 : int            i ;
        31 : {
        32 :    int         j ;
        33 :    char        *ss ;
        34 :
        35 :    j = i ;
        36 :    ss = s ;
```

```
        37 : }
        38 :
        39 : putchar ( c )
        40 : char           c ;
        41 : {
        42 :      char      d ;
        43 :      d = c ;
        44 : }


/*
Target chip : uPD78054
Device file : Vx.xx
*/
```

A.3.3 クロスレファレンス・リスト・ファイル

```
78K/0 Series C Compiler Vx.xx Cross reference List                Date:xx xxx xxxx Page : 1

Command     : -c054 prime.c -a -p -x -e –ng
In-file     : prime.c
Xref-file   : prime.xrf
Para-file   :

ATTRIB MODIFY     TYPE    SYMBOL        DEFINE   REFERENCE

EXTERN            array   mark          5        14      16      22
EXTERN            func    main          7
REG1              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            func    printf        28       18      25
EXTERN            func    putchar       39       20
REG1              pointer s             29       36
PARAM
REG1              int     i             30       35
PARAM
AUTO1             int     j             32       35
AUTO1             pointer ss            33       36
REG1              char    c             40       43
PARAM
AUTO1             char    d             42       43
                  #define TRUE          1        14
                  #define FALSE         2        22
                  #define SIZE          3        5       13      15      21


Target chip :uPD78054
Device file : VX.XX
```

A.3.4 エラー・リスト・ファイル

```
PRIME.C ( 18 ) : CC78K0 warning W0745 : Expected function prototype
PRIME.C ( 20 ) : CC78K0 warning W0745 : Expected function prototype
PRIME.C ( 26 ) : CC78K0 warning W0622 : No return value
PRIME.C ( 37 ) : CC78K0 warning W0622 : No return value
PRIME.C ( 44 ) : CC78K0 warning W0622 : No return value


Target chip : uPD78054
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，CC78K0 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	【オプション指定に関する注意事項】 - 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は省略できません。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	【ファイルの出力先に関する注意事項】 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・メッセージに関する注意事項】 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・ファイル名に関する注意事項】 CC78K0 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，ディフォー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。また，PM plus 使用時は，プライマリ名と拡張子をドット（ . ）で区切り，C ソースの拡張子は，“ .c ”, “ .C ” としてください。 - プライマリ名，拡張子ともに，使用できる文字は，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，PM plus 使用時は，スペース，漢字などの 2 バイト文字を含むパス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（ # ）を使用できません。 - プライマリ名の先頭 8 文字が同一名のファイルは，リンク時にエラーとなります。 - ID78K0/ID78K0-NS/ID78K0-QB，SM78K0 を使用する場合，ファイル名に使える文字は，英小文字（ a ～ z ），英大文字（ A ～ Z ），数字（ 0 ～ 9 ），アンダスコア（ _ ），およびドット（ . ）のみです。
5	【インクルード・ファイルに関する注意事項】 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・ディバ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。

表 B-1 使用上の注意事項

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の注意事項】 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，コンパイラ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。 - 『callf 関数』，『callf 関数以外の関数』の定義は，この 2 種類の定義群をそれぞれまとめて記述してください。 まとめて記述しない場合，アセンブル時にワーニング W2717 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 コンパイラ・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 plus 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，全体オプション指定で -A/-SA オプションを指定してください。 - PM plus 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 RA78K0 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。
7	【利用可能なアセンブラ・パッケージ】 メモリバンクに対応しているため，Ver.3.80 より前の RA78K0 を使用するとエラーになる場合があります。
8	【ネットワーク使用時の注意事項】 一時ファイルを作成するディレクトリをネットワーク上で共有されているファイル・システムに置くと，ご使用になっているネットワーク・ソフトウエアの種類によってはファイルの競合が生じて異常動作を起こす場合があります。オプションや環境変数の設定によって，このような競合を避けてください。 PM plus 使用時は，ネットワーク環境での使用は避けてください。

表 B-1 使用上の注意事項

項番	注意事項
9	【リンク・ディレクティブ・ファイルの作成について】 コンパイラ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，「RA78K0 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「コンパイラに添付されている lk78k0.dr（smp78k0 ディレクトリ以下)」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 @@DATA 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 <lk78k0.dr> memory EXTRAM : (0F000h , 00200h) merge EXTDATA : = EXTRAM リンク・ディレクティブ・ファイル作成時には，次の点をご注意ください。 (1) 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すすめします。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 lk78k0.dr に追加した場合 memory EXTRAM : (0F000h , 00200h) memory STK : (0FB00H , 20H) merge EXTDATA : = EXTRAM （コマンド・ライン） > lk78k0 s0l.rel prime.rel -bcl0.lib -SSTK –Dlk78k0.dr (2) 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・エラーが出ることがあります。 “ RA78K0 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 つの手順によります。 (i) 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)。 (ii) 手順 (i) 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。 (iii) 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 (i) 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。

表 B-1 使用上の注意事項

項番	注意事項
10	【va_start マクロ使用時の注意事項】 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。
11	【SFR（特殊機能レジスタ）定数番地参照時の注意事項】 定数番地参照によって 16 ビット SFR を参照した場合，8 ビット単位でアクセスする不正なコードが生成されますので，SFR 名を使用して参照してください。
12	【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，ご使用の実行形式ファイル（cc78k0.exe，または cc78k0）と同じバージョンで提供されているものをご使用ください。 - 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。
13	【ID78K0-NS, ID78K0, ID78K0-QB におけるソース・ディバグ時の注意事項】 パスカル関数に対する関数コール時，Next コマンドは Step コマンドと同様の動作になります。 Return コマンドなどで呼び出し側の関数に戻ってください。コンパイラ・オプション -ZR を指定した場合は，全関数がパスカル関数となるため，Next コマンドは一切行わないでください。
14	【SM78K0 におけるソース・ディバグ時の注意事項】 パスカル関数に対する関数コール時，Next コマンドを行うと暴走するので，Next コマンドを行わないでください。 コンパイラ・オプション -ZR を指定した場合は，全関数がパスカル関数となるため，Next コマンドは一切行わないでください。

表 B-1 使用上の注意事項

項番	注意事項
15	【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。CC78K0 では，ディフォールトで 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。 C コンパイラ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，表 8-4 参照してください。 スタートアップ・ルーチン： (1) C 標準ライブラリ用の領域を使用しない場合：s0.rel (2) C 標準ライブラリ用の領域を使用する場合：s0l.rel 【使用例】 C:>lk78k0 s0.rel sample.rel -S -Bcl0.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 s0.rel ：スタートアップ・ルーチン cl0.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 - 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。 - ユーザがライブラリを作成する場合は，CC78K0 が提供するライブラリとは分けて作成し，リンク時にコンパイラのライブラリよりも前に指定するようにしてください。 - CC78K0 のライブラリに，ユーザ関数を追加しないでください。 - 浮動小数点ライブラリ（cl0*f.lib）を使用する場合は，通常のライブラリ（cl0*.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 -Bcl0f.lib -Bcl0.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 -Bcl0.lib -Bcl0f.lib
16	【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】 CC78K0 では，ユーザはスタック領域を確保できません。 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 plus 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。
17	【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（指定の解除をしないでください）。 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。
18	【ヘルプ指定オプションについて】 PM plus 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。

表 B-1 使用上の注意事項

項番	注意事項
19	【-LL オプション指定について】 PM plus 使用時は，-LL オプションに指定できる最大の数字は，32767 とします。32768 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。
20	【シンボル名長に関する注意事項】 ID78K0-NS V2.01，SM78K0 V2.10 以前を使用する場合は，127 文字を越えるシンボル名を使用しないでください。
21	【PM plus 使用時の注意事項】 (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 plus に指定すると，PM plus 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時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 plus 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〉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〉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いませんので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 plus 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なしであると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 plus の「ProjectWindow」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 コンパイラ付属のスタートアップ・ルーチン，標準ライブラリは，PM plus の [プロジェクト] メニュー，およびプロジェクト・ウインドウからの右クリックの “ プロジェクト関連ファイルの追加 ” から追加，削除できません。 コンパイラ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。

表 B-1 使用上の注意事項

<table>
<tr><th>項番</th><th>注意事項</th></tr>
<tr><td>22</td><td>
【プロトタイプ宣言に関する注意事項】
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
<pre>f (void) ;   /* E0301 : Syntax error */
                    /* E0701 : External definition syntax */</pre>

【回避策】 関数の型を記述してください。

例
<pre>int   f (void) ;</pre>
</td></tr>
<tr><td>23</td><td>
【エラー・メッセージ出力に関する注意事項】
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
<pre>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 */</pre>

【回避策】 ありません。
</td></tr>
<tr><td>24</td><td>
【前処理指令中のコメント記述に関する注意事項】
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
<pre>/* 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 */</pre>

【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
<pre>#pragma   sfr   /* com1 */
#define   ONE   1   /* com2 */
#define   TWO   2   /* com3 */
#ifdef   ANSI_C   /* com4 */

#endif   /* com5 */
#define   SUB (p1 , p2) p2 = p1   /* com6 */</pre>
</td></tr>
</table>

表 B-1 使用上の注意事項

項番	注意事項

25	**【構造体 / 共用体 /enum のタグ使用に関する注意事項】**
	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 ) {`
	`func ( sizeof ( st ) / sizeof ( st [ 0 ] ) , st ) ; /* W0510 Pointer mismatch */`
	`}`
	(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
	`void func1 ( int , struct st ) ; /* E0737 Undeclared structure/union/enum tag */`
	`struct st func2 ( int ) ; /* E0737 Undeclared structure/union/enum tag */`
	`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。**

26	**【関数内の配列 / 構造体 / 共用体の初期化に関する注意事項】**
	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 ;`
	`}`

表 B-1 使用上の注意事項

項番	注意事項

27	【extern callt 関数に関する注意事項】
	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
	} ;
	callf 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。

28	【構造体を返す関数に関する注意事項】
	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。

表 B-1 使用上の注意事項

項番	注意事項

29	**【共用体の初期化に関する注意事項】**
	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 } ;

30	**【漢字コード種別に関する注意事項】** 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。**

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	なし
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）	メモリの割り付け方法を指定します。	-R と -NR，-RD と -NR，-RK 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。		
	-RK [n] [M]（n = 1 , 2 , 4 ）	関数引数，および auto 変数（static auto 変数を除く）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 , -RK , -RSオプションを無効にします。		
最適化指定	-Q [最適化種別]（複数の種別を指定する場合は続けて指定します）	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。		
ディバグ情報出力指定	-G [n]（ n = 1, 2 ）	ソース・レベルのディバ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。		

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
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 ディレクトリ [, ディレクトリ] …（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ディレクトリから入力するよう指定します。	独立	1. ソース・ファイルのあるディレクトリ 2. 環境変数 INC78K0 により指定されたディレクトリ 3. C:\NECTools32\inc78k0
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	
クロスレファレンス・リスト・ファイル作成指定	-X [出力ファイル名]	クロスレファレンス・リスト・ファイルの出力を指定します。	独立	クロスレファレンス・リスト・ファイルを出力しません。

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
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	-LL66（コンソール出力の場合は 65535 文字とします）
	-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	なし
	-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	なし
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，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・ファイル作成ディレクトリ指定	-T ディレクトリ	テンポラリ・ファイルを指定したドライブ，ディレクトリに作成します。	独立	環境変数 TMP により指定されたドライブ，ディレクトリ
ヘルプ指定	-- / -? / -H	--/-?/-H オプションは，オプションの簡単な説明，ディフォールト・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

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
機能拡張指定	-Z [種別] (複数の種別を指定する場合は続けて指定します)	種別指定に対応する処理を指定します。	-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 ディレクトリ	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
スタティック・モデル指定	-SM [n] (n = 1-16)	オブジェクトのスタティック・モデルかノーマルモデルを指定します。	独立	ノーマル・モデル (n = 0)
関数情報ファイル指定	-MF ファイル名	64K バイトの空間を越えるコード部への配置が可能となります。	独立	共通領域配置

付録 D　総合索引

S

T

U

V

W

X

Y

Z

【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。