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

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

CC78K4　Ver. 2.40以上

Cコンパイラ

操作編

対象デバイス
78K4シリーズ

資料番号　U16707JJ1V0UM00（第1版）
発行年月　June 2003 NS　CP(K)

〔メ　モ〕

目次要約

Windows**および**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。**

SPARCstation**は，米国**SPARC International, Inc.**の商標です。**

SunOS, Solaris**は，米国サン・マイクロシステムズ社の商標です。**

HP9000**シリーズ**700, HP-UX**は，米国**Hewlett-Packard Corp.**の商標です。**

- 本資料に記載されている内容は2003年6月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品の使用に関連し発生した第三者の特許権，著作権その他の知的財産権の侵害等に関し，一切その責を負いません。当社は，本資料に基づき当社または第三者の特許権，著作権その他の知的財産権を何ら許諾するものではありません。
- 本資料に記載された回路，ソフトウエアおよびこれらに関する情報は，半導体製品の動作例，応用例を説明するものです。お客様の機器の設計において，回路，ソフトウエアおよびこれらに関する情報を使用する場合には，お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し，当社は，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等安全設計を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「特定水準」に分類しております。また，各品質水準は，以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認ください。

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

  特別水準：輸送機器（自動車，電車，船舶等），交通用信号機器，防災・防犯装置，各種安全装置，生命維持を目的として設計されていない医療機器

  特定水準：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等

当社製品のデータ・シート，データ・ブック等の資料で特に品質水準の表示がない場合は，標準水準製品であることを表します。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

(注)

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

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

M8E 02.11

はじめに

このマニュアルは，CC78K4（78K4**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，CC78K4**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，“**CC78K4　C**コンパイラ　ユーザーズ・マニュアル　言語編（**U15556J**）”（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】

CC78K4**では，**78K4**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，**RA78K4（78K4**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　CC78K4**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，使用上の注意事項一覧，CC78K4**に関する制限事項一覧，および索引があります。**

【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。

資　料　名		資料番号	
		和文	英文
CC78K4　Cコンパイラ	操作編	このマニュアル	U16707E
	言語編	U15556J	U15556E
RA78K4　アセンブラ・パッケージ	操作編	U16708J	U16708E
	言語編	U15255J	U15255E
	構造化アセンブリ言語編	U11743J	U11743E
SM78Kシリーズ　システム・シミュレータ　Windows®ベース	操作編	作成予定	作成予定
SM78Kシリーズ　システム・シミュレータ　Ver.1.40以上	外部部品ユーザ・オープン・インタフェース仕様編	U10092J	U10092E
ID78K4-NS　統合ディバッガ　Ver.2.52以上 Windowsベース	操作編	U16632J	U16632E
ID78K4　統合ディバッガ　Windowsベース	レファレンス編	U10440J	U10440E
RX78K4　リアルタイムOS	基礎編	U10603J	U10603E
	インストール編	U10604J	U10604E
	ディバッガ編	U10364J	－
PM plus Ver.5.10		U16569J	作成予定

【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：78K4**シリーズ用 リアルタイム**OS　RX78K4

…　：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¥smp78k4¥cc78k4¥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

第1章　概　　説

CC78K4シリーズ　Cコンパイラは，ANSI-C[注]または78K4シリーズのC言語で記述された，Cソース・プログラムを78K4シリーズ用の機械語に変換するプログラムです。

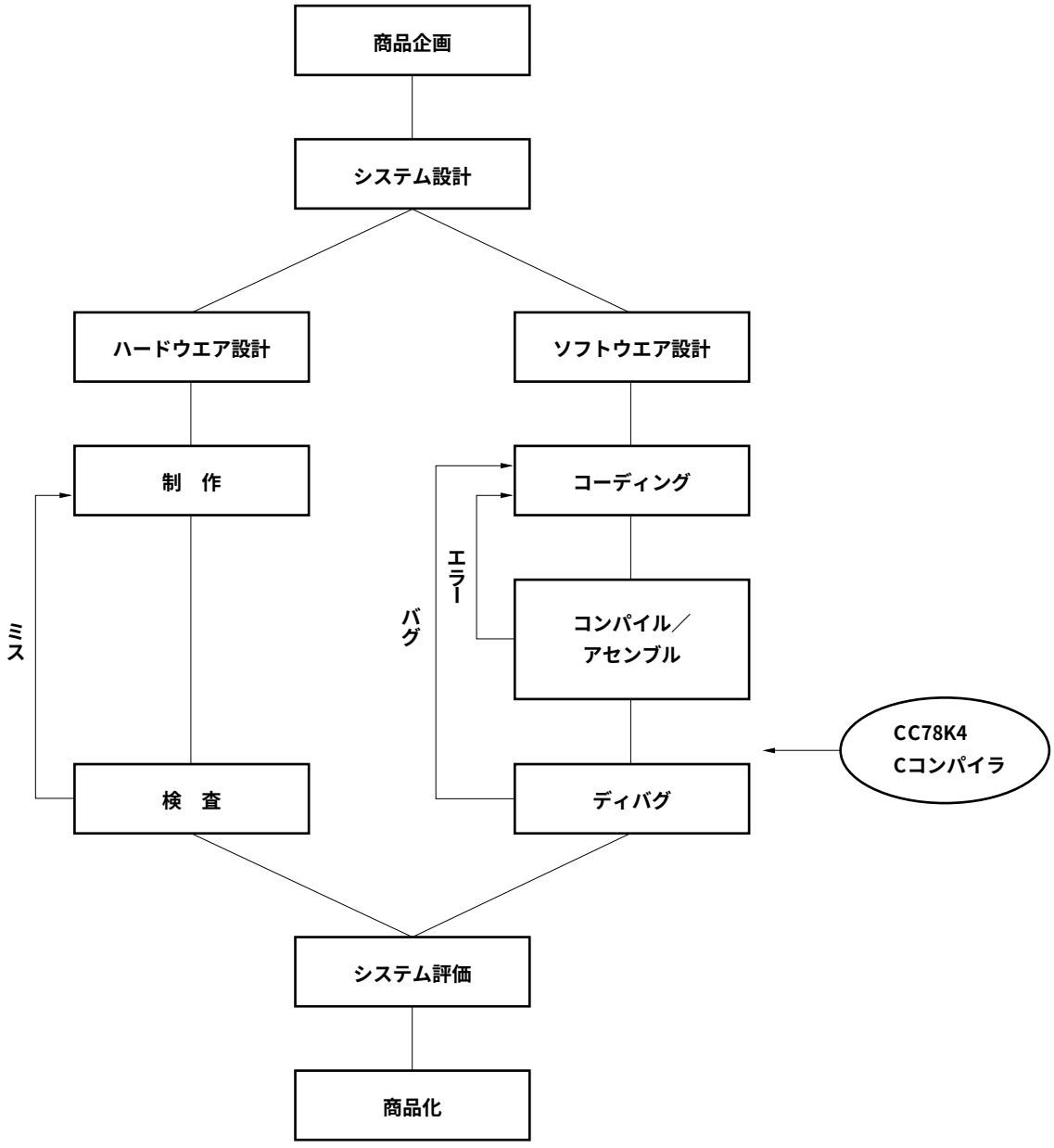
本Cコンパイラは，78K4シリーズ用アセンブラ・パッケージに添付されているPM plusを使用した場合に，Windows98/Me/2000/XP，またはWindowsNT™4.0上で起動できるようになっています。PM plusを使用しない場合は，DOSプロンプト（Windows98/Me）またはコマンド・プロンプト（WindowsNT4.0/2000/XP）上で起動します（Windows版の場合）。

注　ANSI-Cとは，American National Standards Instituteの規格に準拠しているC言語のことです。

1. 1　マイクロコンピュータ応用製品の開発とCC78K4の役割

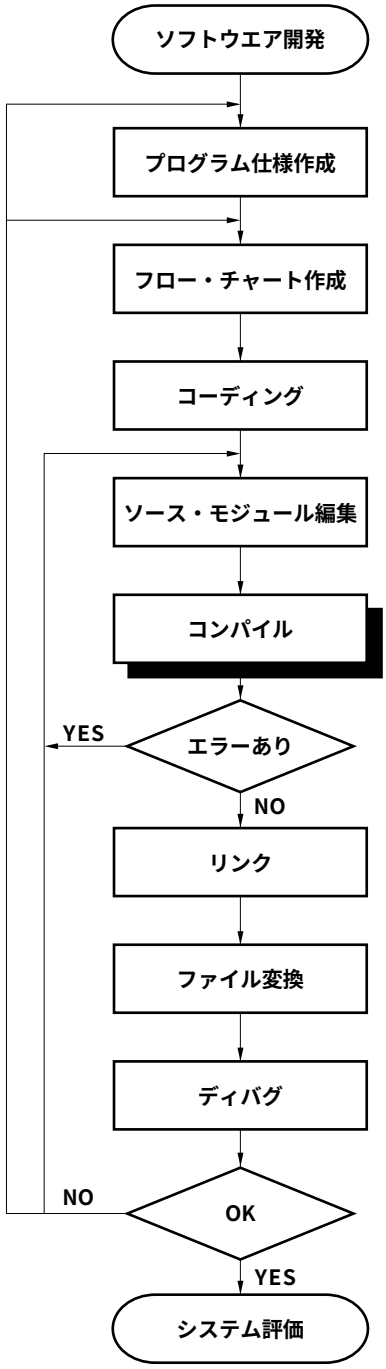
製品開発における，CC78K4の位置づけを次に示します。

図1－1　マイクロコンピュータ応用製品の開発工程



ソフトウエア開発の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，関数ライブラリと結合します。

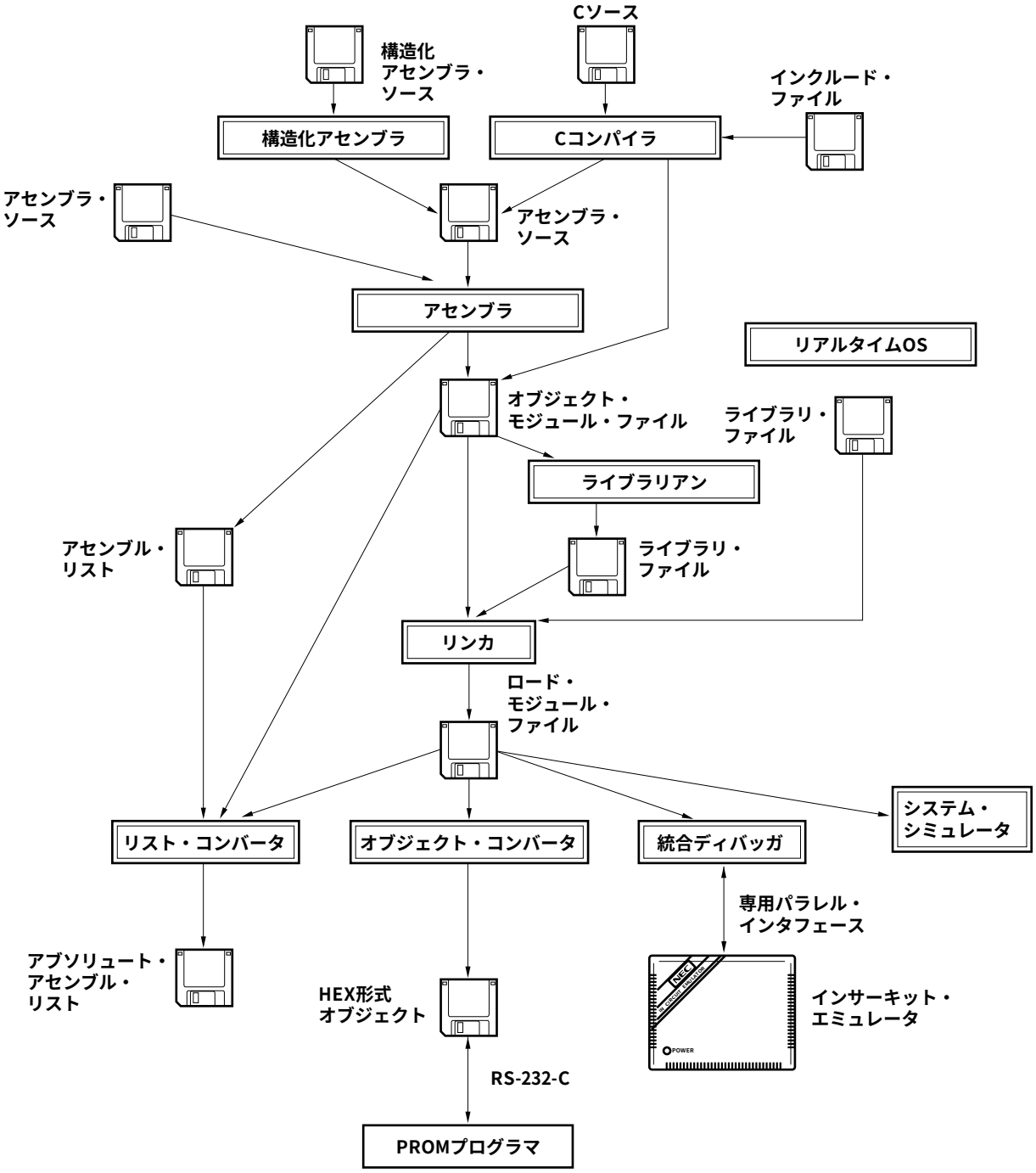
・・・ファイルをHEX形式に変換します。

・・・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　CC78K4による開発手順

CC78K4における開発手順を次に示します。

図1－3　CC78K4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

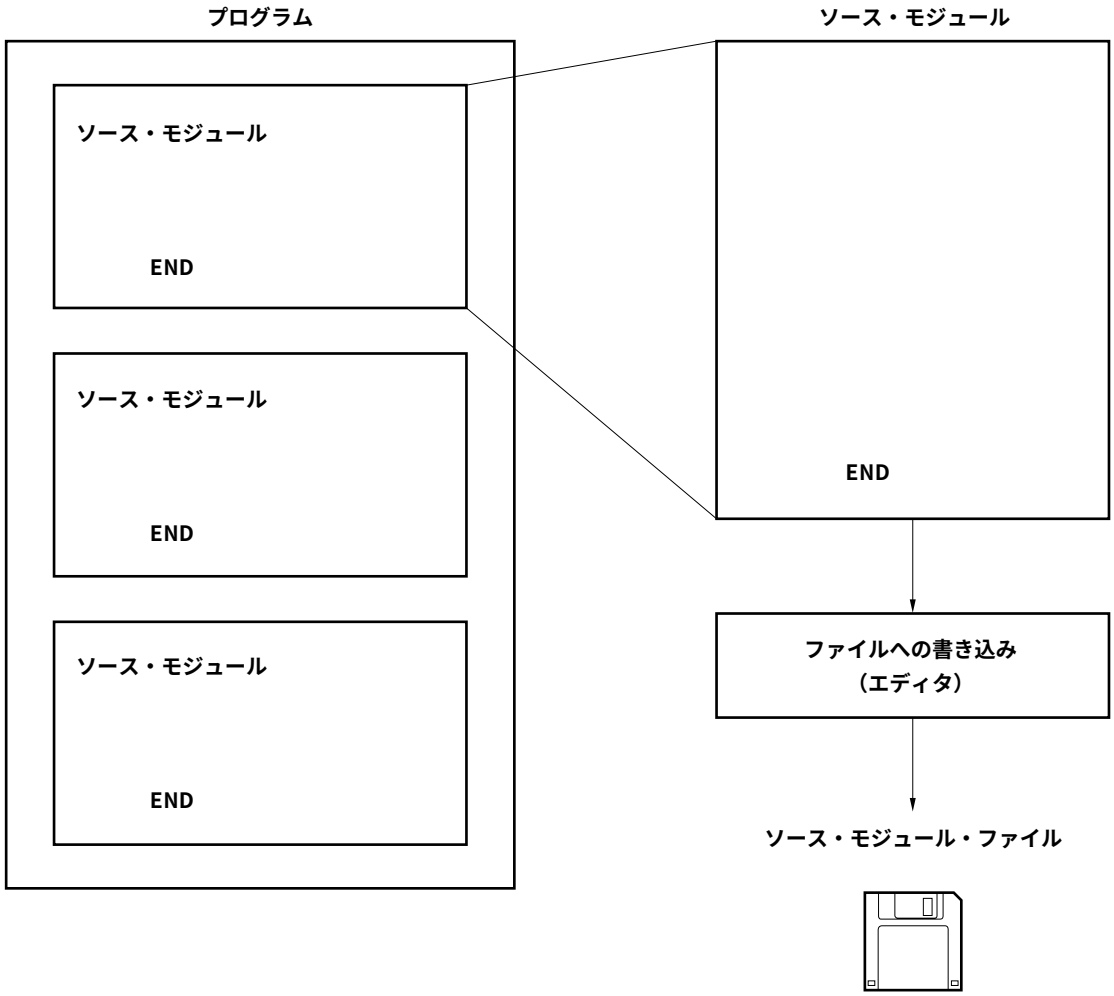
1つのプログラムを機能的にいくつかに分割します。

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，CC78K4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

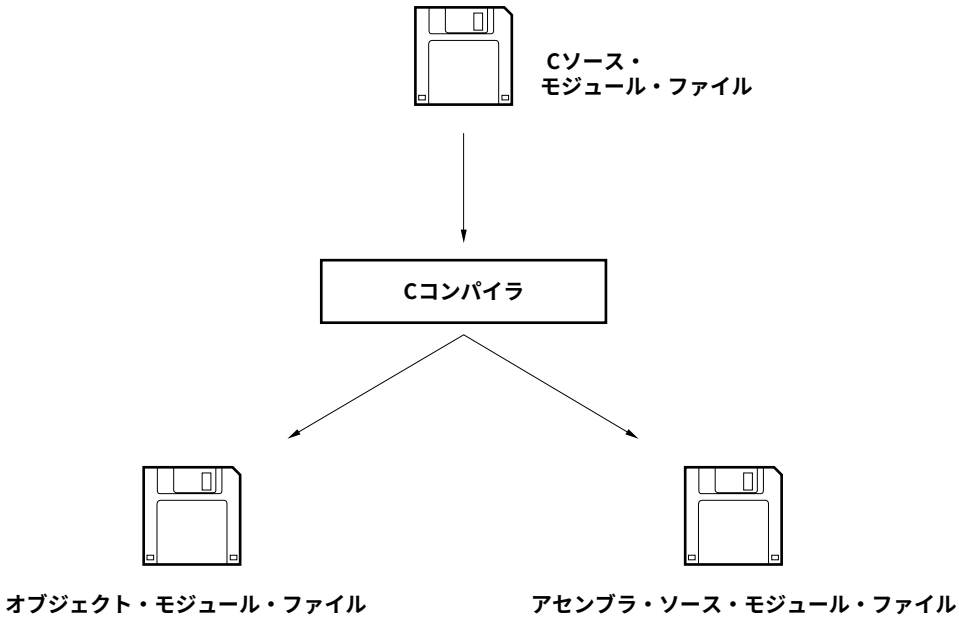


1. 2. 2　Cコンパイラ

Cコンパイラは，Cソース・モジュールを入力し，C言語を機械語に変換します。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・エラーがない場合には，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プログラムの修正，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。出力したい場合には，コンパイルの際にアセンブラ・モジュール・ファイル作成指定の-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ー5　Cコンパイラの機能



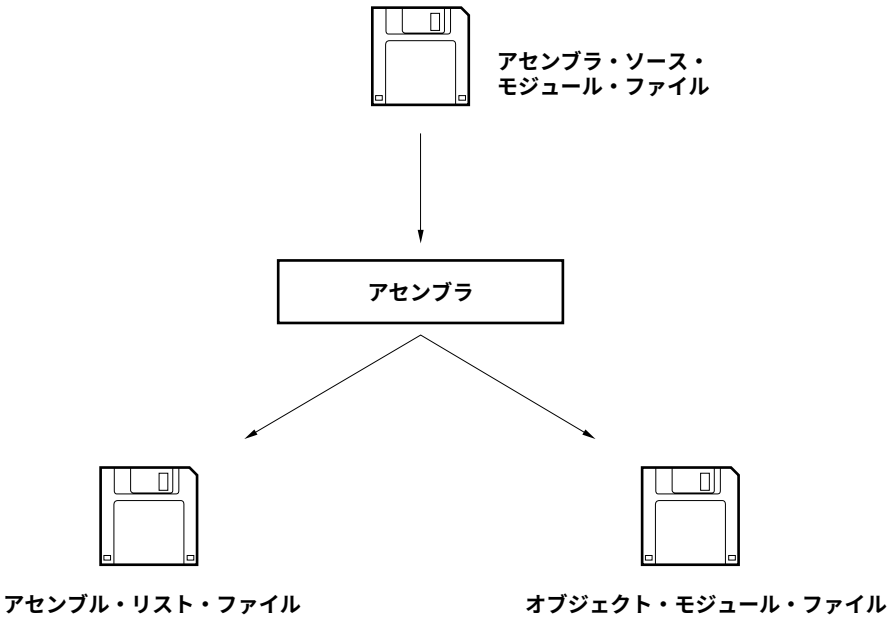
1. 2. 3　アセンブラ

アセンブルは，RA78K4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使用して行います。

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，アセンブリ言語を機械語に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・エラーがない場合には，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と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



1. 2. 4 リ ン カ

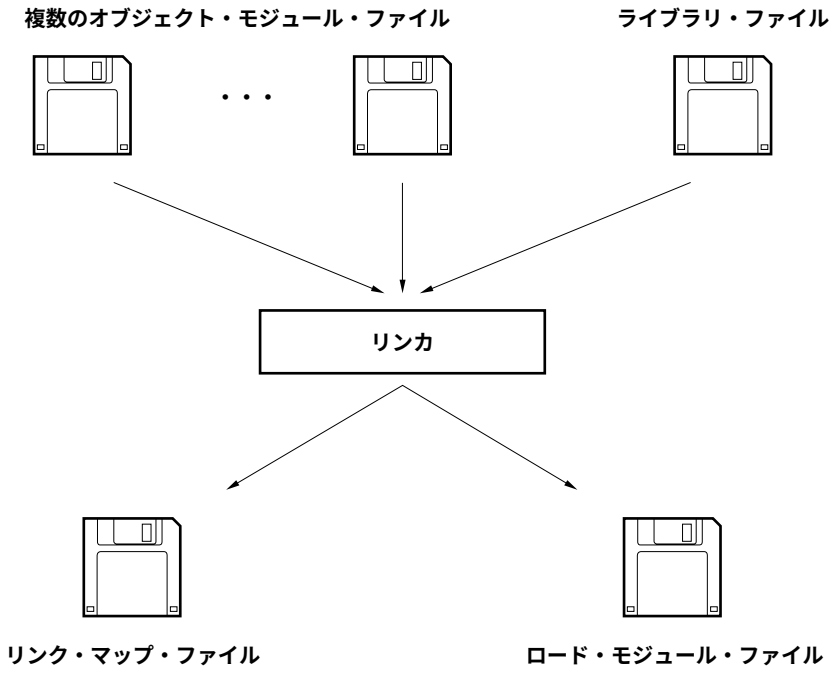
リンクは，RA78K4アセンブラ・パッケージ（別売）に含まれているリンカを使用して行います。

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・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

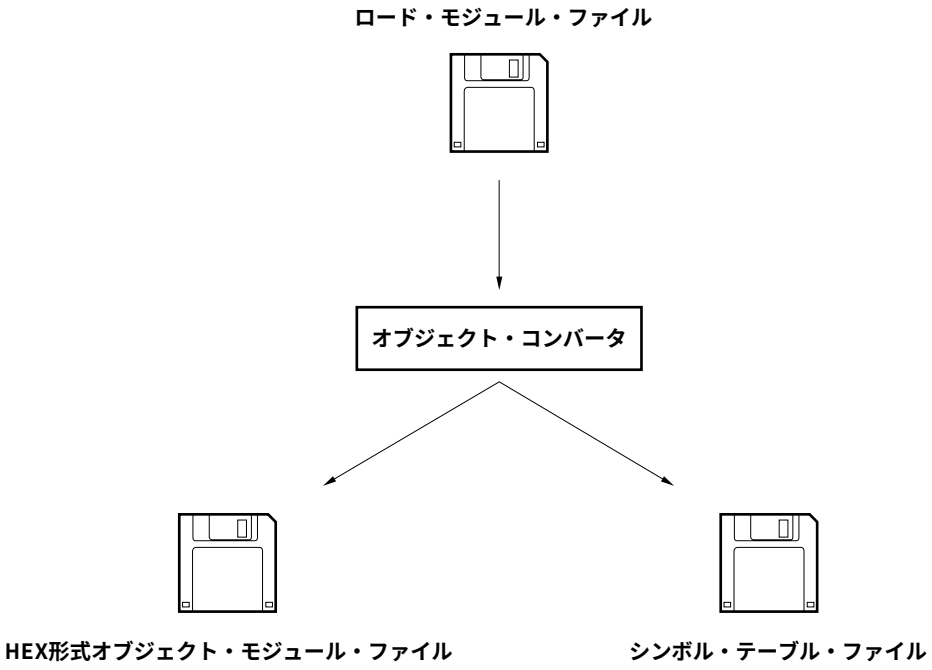


1. 2. 5　オブジェクト・コンバータ

オブジェクト・コンバータは，RA78K4 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



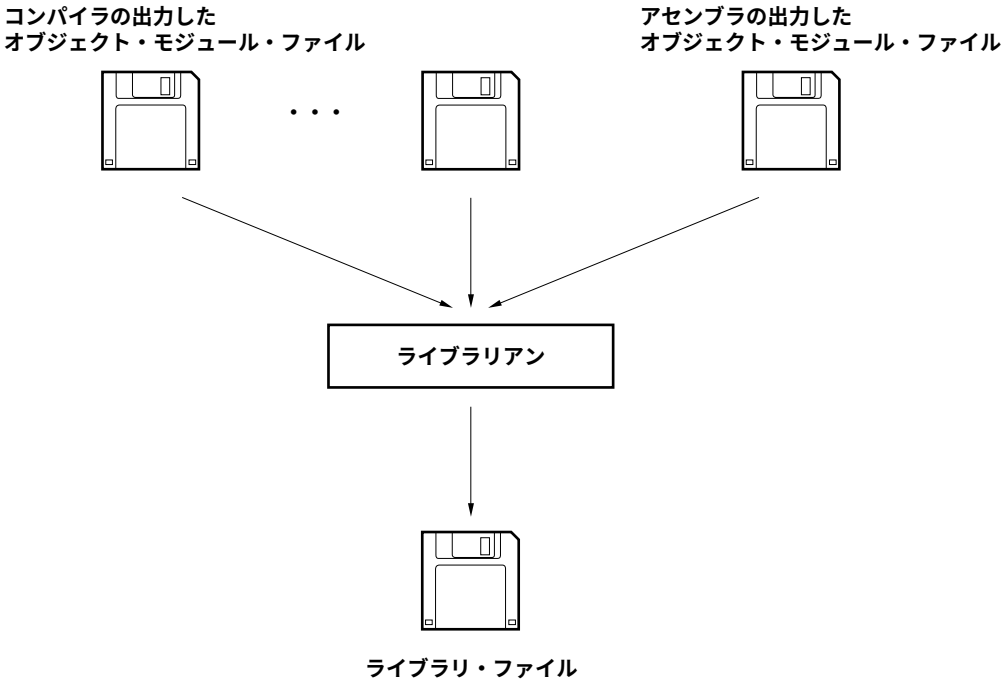
1. 2. 6　ライブラリアン

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，RA78K4 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

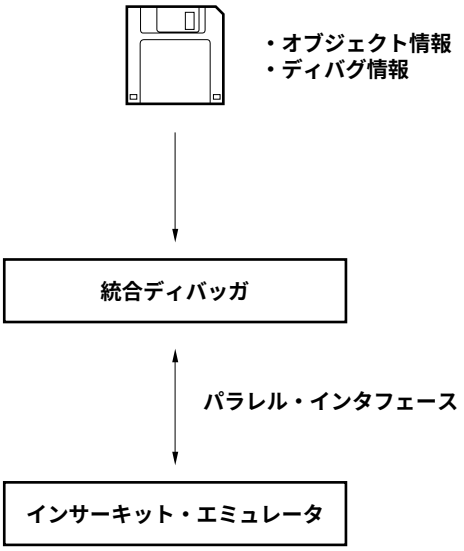


1. 2. 7　ディバッガ

リンカが出力したロード・モジュール・ファイルをID78K4, ID78K4-NS（78K4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。

図1－10　ディバッガの機能

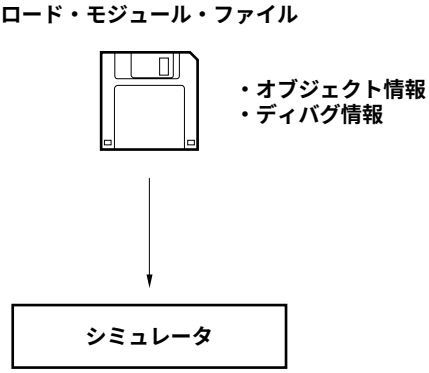


1. 2. 8　システム・シミュレータ

リンカが出力したロード・モジュール・ファイルをSM78K4（78K4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・インターフェースを用いたソース・ディバグが可能となります。

SM78K4は，ID78K4, ID78K4-NS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。SM78K4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。

図1－11　シミュレータの機能

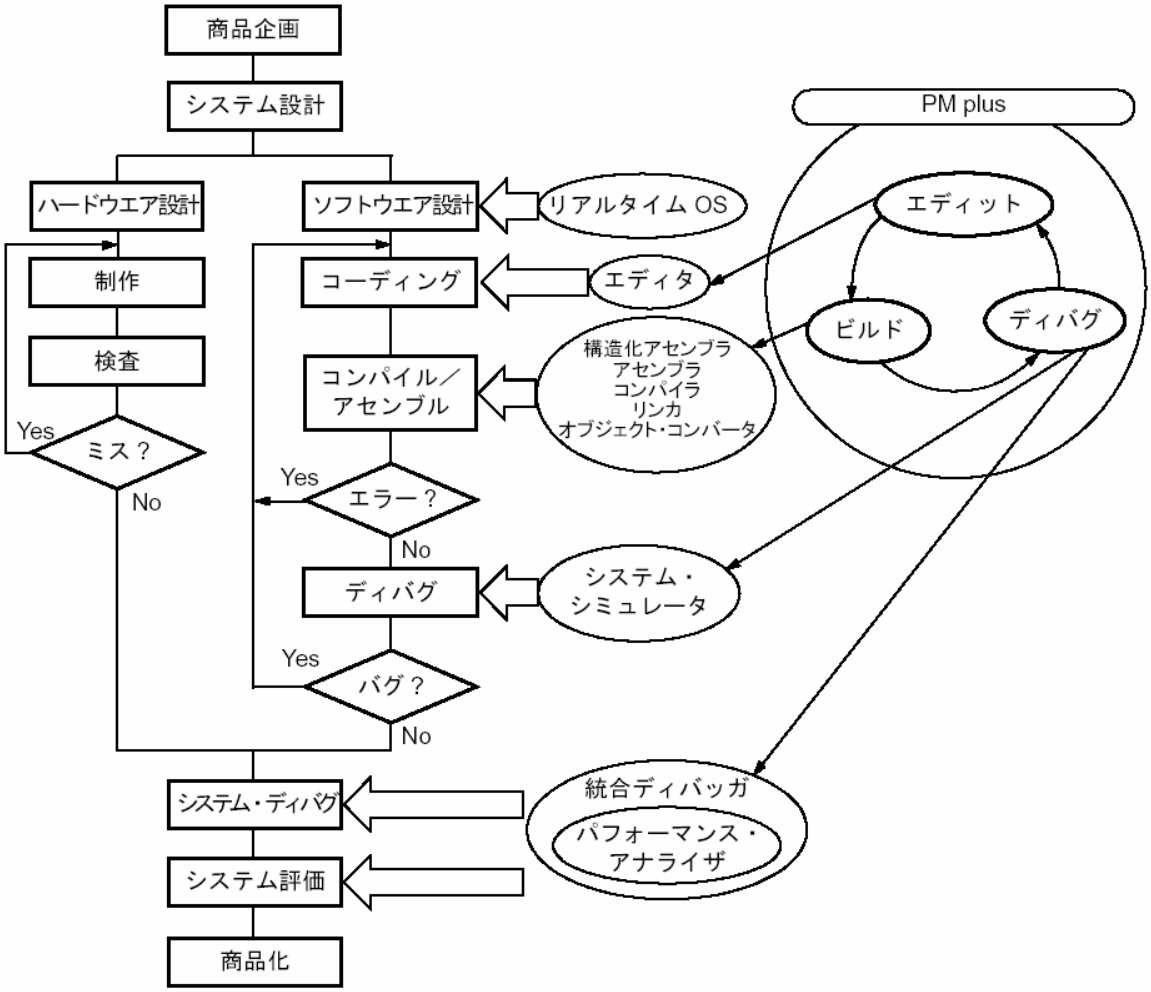


1. 2. 9　PM plus

PM plusは，CC78K4に添付されているDLLファイルを使用し，CC78K4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，RA78K4　アセンブラ・パッケージに添付されます。インストールと設定は，RA78K4　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。CC78K4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，RA78K4　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　ビルドとは，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，CC78K4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	記録形式
PC-9800シリーズ	日本語Windows（98/Me/2000/XP/NT4.0）注	CD-ROM	Windows標準のインストーラ対応
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		
HP9000シリーズ700™	HP-UX™（Rel.10.20以降）	CD-ROM	cpコマンド
SPARCstation™ファミリ	SunOS™（Rel.4.1.4以降） Solaris™（Rel.2.5.1以降）		

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

2. 2. 1　Windows版のインストール

以下に，CC78K4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

CC78K4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 CC78K4￥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，CC78K4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. 1　Windows**版のディレクトリ構成**を参照してください。

2. 2. 2　UNIX版のインストール

以下の手順でインストールしてください。ここでは/nectoolsにインストールするものと仮定します。

(1) ログイン

ホスト・マシンにログインします。

(2) ディレクトリの移動

インストール・ディレクトリへ移動します。

```
% cd　/nectools
```

(3) 提供媒体のセット

CD-ROMをCDドライブにセットし，CDドライブを閉じます。

(4) cp**コマンドを実行し，**CD-ROM**から各種ファイルをコピーします。（**CD-ROM**がマウントされていることを確認してからコピーしてください。）**

(5) 環境変数PATH**に**/nectools/bin**を追加します。**

2. 3　デバイス・ファイルのインストール

2. 3. 1　Windows版のインストール

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，CC78K4とともにインストールされます。

2. 3. 2　UNIX版のインストール

デバイス・ファイルは，-yオプションにより，ディレクトリを指定するか（例:-y/nectools/dev），コンパイラの実行形式のあるディレクトリ（例：/nectools/bin）にコピーしてご使用ください。

2. 4　ディレクトリ構成

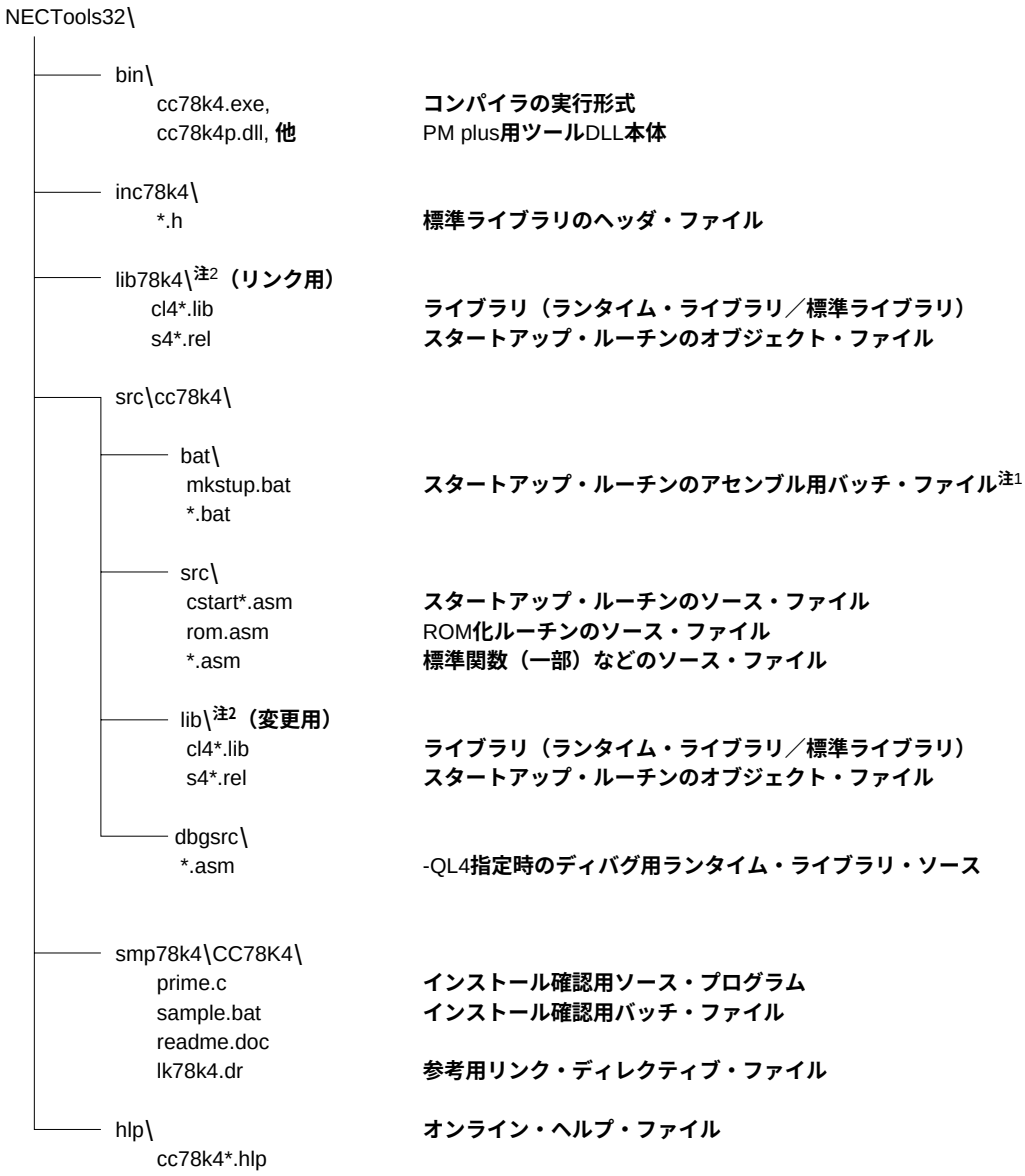
2. 4. 1　Windows版のディレクトリ構成

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（CC78K4, RA78K4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　Windows版のディレクトリ構成



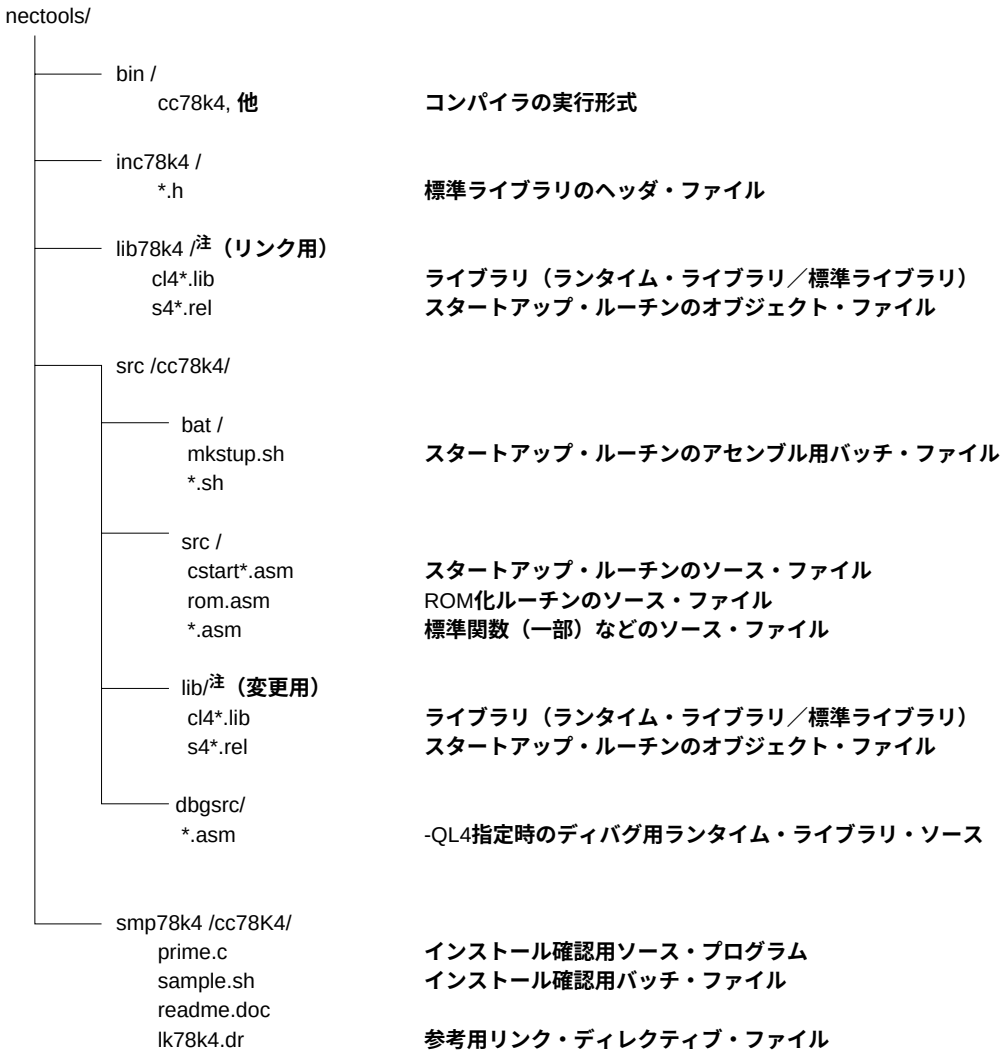
注1．**このバッチ・ファイルは，**PM plus**上では使用できません。バッチ・ファイルを使用する場合は，**DOS**プロンプト（**Windows98/Me**）またはコマンド・プロンプト（**Windows2000/XP/NT4.0**）で実行してください。**

2．lib78k4**ディレクトリ以下のスタートアップ・ルーチン，ライブラリは，**src¥cc78k4**ディレクトリ以下のものとまったく同じものですが，スタートアップ・ルーチンを修正する場合は，**src¥cc78k4**ディレクトリ以下のソースを修正し，バッチ・ファイルでアセンブルしたものが**src¥cc78k4¥lib**に入るので，**lib78k4**ディレクトリにコピーしてリンクしてください。**

2. 4. 2　UNIX**版のディレクトリ構成**

cp**コマンドで**/nectools**にインストールした場合のディレクトリ構成は，次のようになります。**

図2ー2　UNIX**版のディレクトリ構成**



注　lib78k4**ディレクトリ以下のスタートアップ・ルーチン，ライブラリは，**src/cc78k4**ディレクトリ以下のものとまったく同じものですが，スタートアップ・ルーチンを修正する場合は，**src/cc78k4**ディレクトリ以下のソースを修正し，バッチ・ファイルでアセンブルしたものが**src/cc78k4/lib**に入るので，**lib78k4**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

2.5.1　Windows版のアンインストール

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

(1) Windows の起動

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

(2) <コントロールパネル>ウィンドウのオープン

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>ウィ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のプロパティ>ウィンドウのオープン

<コントロールパネル>ウィンドウの[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のプロパティ>ウィンドウをオープンします。

(4) CC78K4 の削除

<アプリケーションの追加と削除のプロパティ>ウィンドウの≪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 CC78K4 78K/IV Cコンパイラ Vx.xx”を選択した後，追加と削除(R)... ボタンをクリックします。

なお，<ファイル削除の確認>ウィンドウがオープンした際には，はい(Y) ボタンをクリックします。

(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，2.4.1 Windows版のディレクトリ構成を参照してください。

2.5.2　UNIX版のアンインストール

2.2.2　UNIX版のインストールでコピーしたファイルを，rmコマンドで削除します。

2. 6　環境設定

2. 6. 1　ホスト・マシン（PC-9800シリーズ，IBM PC/AT互換機の場合）

CC78K4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

EWSおよび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	コンパイラの実行形式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 （PC-9800シリーズ, IBM PC/AT互換機のみ有効）
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（PC-9800シリーズ，IBM PC/AT互換機，HP9000シリーズ700のディフォールト） euc　EUC（SPARCstationのディフォールト） none　2バイト・コードなし
INC78K4	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（EWS版のみ必要）
LIB78K4	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（EWS版のみ必要）

【指 定 例】

PC-9800シリーズ，IBM PC/AT互換機の場合

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

HP9000シリーズ700，SPARCstationの場合

cshを使用している場合の例

```
set path = ($path /nectools/bin)
setenv LANG78K    euc
setenv INC78K4    /nectools/inc78k4
setenv LIB78K4    /nectools/lib78k4
```

shを使用している場合の例

```
PATH = $PATH:/nectools/bin
LANG78K = euc
INC78K4 = /nectools/inc78k4
LIB78K4 = /nectools/lib78k4
export PATH LANG78K INC78K4 LIB78K4
```

2. 6. 3　ファイル構成

各ディレクトリの内容を次の表に示します。ここでは，PC-9800シリーズ，IBM PC/AT互換機用のファイルについて説明します。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。

備考　UNIX用のものは，ファイルの拡張子が一部異なります。

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¥	cc78k4.exe	コンパイラ本体
	cc78k4.msg	メッセージ・ファイル
	＊.hlp	ヘルプ・ファイル
	＊.dll	DLLファイル
INC78K4¥	＊.h[注1]	標準ライブラリのヘッダ・ファイル
SRC¥CC78K4¥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¥CC78K4¥SRC	cstart＊.asm[注4]	スタートアップ・ルーチンのソース・ファイル
	rom.asm	ROM化ルーチンのソース・ファイル
	＊.asm[注5]	標準関数（一部）などのソース・ファイル
SRC¥CC78K4¥DBGSRC	＊.asm	-QL4指定時のディバグ用ランタイム・ライブラリ・ソース
HLP	＊.hlp	オンライン・ヘルプ・ファイル

注1.　**言語編**（U15556J）10. 2　**ヘッダ・ファイル**を参照してください。

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。

3.　**表**8－1　**ディレクトリ“**BAT**”の内容**を参照してください。

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）

5.　**表**8－2　**ディレクトリ“**SRC**”の内容**を参照してください。

2. 6. 4　ライブラリ・ファイル

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	
LIB78K4¥	CL4S.LIB	CL4S.LIB	CL4SE.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]
	CL4SO.LIB	CL4SO.LIB	CL4SOE.LIB	
	CL4SR.LIB	CL4SR.LIB	CL4SRE.LIB	
	CL4.LIB	CL4.LIB	CL4E.LIB	
	CL4M.LIB	CL4M.LIB	CL4ME.LIB	
	CL4MR.LIB	CL4MR.LIB	CL4MRE.LIB	
	CL4O.LIB	CL4O.LIB	CL4OE.LIB	
	CL4OP.LIB	CL4OP.LIB	CL4OPE.LIB	
	CL4P.LIB	CL4P.LIB	CL4PE.LIB	
	CL4PR.LIB	CL4PR.LIB	CL4PRE.LIB	
	CL4R.LIB	CL4R.LIB	CL4RE.LIB	
	CL4SF.LIB	CL4SF.LIB	CL4SFE.LIB	
	CL4SFR.LIB	CL4SFR.LIB	CL4SFRE.LIB	
	CL4F.LIB	CL4F.LIB	CL4FE.LIB	
	CL4FR.LIB	CL4FR.LIB	CL4FRE.LIB	
	CL4MF.LIB	CL4MF.LIB	CL4MFE.LIB	
	CL4MFR.LIB	CL4MFR.LIB	CL4MFRE.LIB	
	S4.REL	S4B.REL	S4E.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]
	S4C.REL	S4CB.REL	S4CE.REL	
	S4CL.REL	S4CLB.REL	S4CLE.REL	
	S4CLP.REL	S4CLPB.REL	S4CLPE.REL	
	S4CP.REL	S4CPB.REL	S4CPE.REL	
	S4L.REL	S4LB.REL	S4LE.REL	
	S4LP.REL	S4LPB.REL	S4LPE.REL	
	S4M.REL	S4MB.REL	S4ME.REL	
	S4MC.REL	S4MCB.REL	S4MCE.REL	
	S4MCL.REL	S4MCLB.REL	S4MCLE.REL	
	S4ML.REL	S4MLB.REL	S4MLE.REL	
	S4P.REL	S4PB.REL	S4PE.REL	
	S4S.REL	S4SB.REL	S4SE.REL	
	S4SL.REL	S4SLB.REL	S4SLE.REL	

注1.　**ライブラリの命名規則は，次のようになっています。**

```
cl4<model><i/f><align><float><pascal><flash>.lib
```

<model>

なし　**ラージ・モデル（コンパイル・オプション**-ML**指定時）**

m　**ミディアム・モデル（コンパイル・オプション**-MM**指定時）**

s　**スモール・モデル（コンパイル・オプション**-MS**指定時）**

<i/f>

なし　CC78K4 V2.00**仕様の関数インタフェースを使用（コンパイル・オプション**-ZO**を指定しないとき）**

o　CC78K4 V1.00**仕様の関数インタフェースを使用（コンパイル・オプション**-ZO**指定時）**

備考　**ラージ／スモール・モデルのときのみ有効です。ミディアム・モデルの場合はなしです。**

<align>

なし　**コンパイル・オプション**-RP/-RA**を指定しない場合**

p　**コンパイル・オプション**-RP/-RA**を指定した場合**

備考　**ラージ・モデルのときのみ有効です。スモール／ミディアム・モデルの場合はなしです。**

<float>

なし　**標準ライブラリ，ランタイム・ライブラリ（浮動小数点数未使用）**

f　**浮動小数点数ライブラリ用**

<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）**

<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。**

```
s4<model><loc><lib><align><flash>.rel
```

<model>**（メモリ・モデル）**

なし	**ラージ・モデル（コンパイル・オプション**-ML**指定時）**
m	**ミディアム・モデル（コンパイル・オプション**-MM**指定時）**
s	**スモール・モデル（コンパイル・オプション**-MS**指定時）**

<loc>

なし	**コンパイル・オプション**-CS15**指定時**
c	**コンパイル・オプション**-CS0**指定時**

備考　**ラージ／ミディアム・モデルのときのみ有効です。スモール・モデルの場合はなしです。**

<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**

<align>

なし	**コンパイル・オプション**-RP/-RA**を指定しない場合**
p	**コンパイル・オプション**-RP/-RA**を指定した場合**

備考　**ラージ・モデルのときのみ有効です。スモール／ミディアム・モデルの場合はなしです。**

<flash>

なし	**通常**
b	**ブート領域用**
e	**フラッシュ領域用**

第3章　コンパイルからリンクまでの手順

この章では，CC78K4とRA78K4　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，PC-9800シリーズ版およびIBM PC/AT互換機版については，PM plus上で実行する方法を，その他については，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，RA78K4　アセンブラ・パッケージに含まれるPM plusでCC78K4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CC78K4を起動する場合，CC78K4に含まれるCC78K4P.DLLが参照されます。

3. 1. 1　CC78K4P.DLL（ツールDLL）の位置づけ

CC78K4P.DLLファイルなどのツールDLLファイルは，Windows版の78K4シリーズ用Cコンパイラ（CC78K4）をWindows98/Me/2000/XPまたはWindowsNT4.0上で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。

製品（英語Windows／日本語Windows版）の違いにより，日本語表示・英語表示モードが切り替わります。

3. 1. 3　CC78K4用オプション設定メニュー

(1) オプション・メニュー項目

CC78K4　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］メニュー中に次の項目が追加されます。

日本語版：［コンパイラオプションの設定（C）...］
英語版　：［Compiler Options...］

(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］ → ［コンパイラオプションの設定］ メニューを選択すると，ツールDLLのオプション設定機能が呼ばれます。

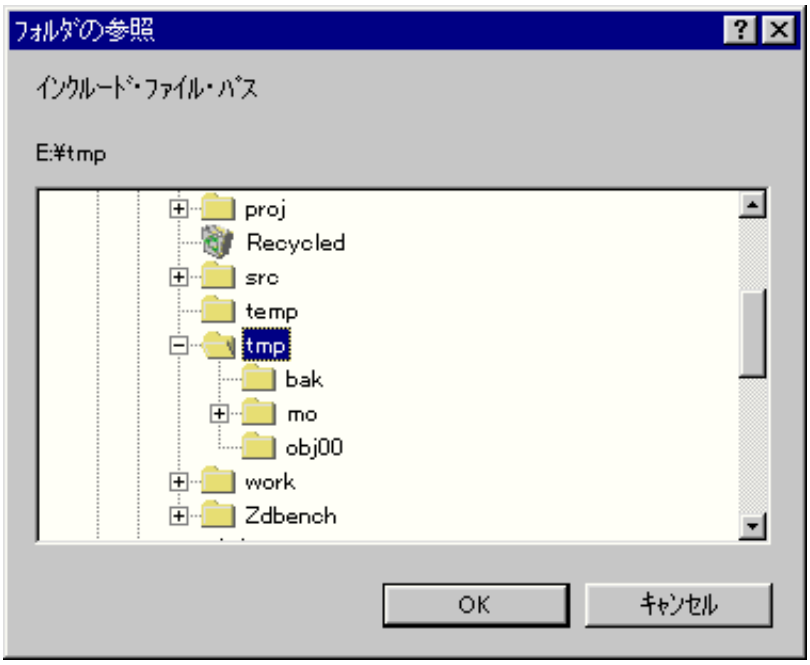
＜コンパイラオプションの設定＞ダイアログは次のとおりです。

(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。

- インクルード・ファイル・パス
- 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



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。

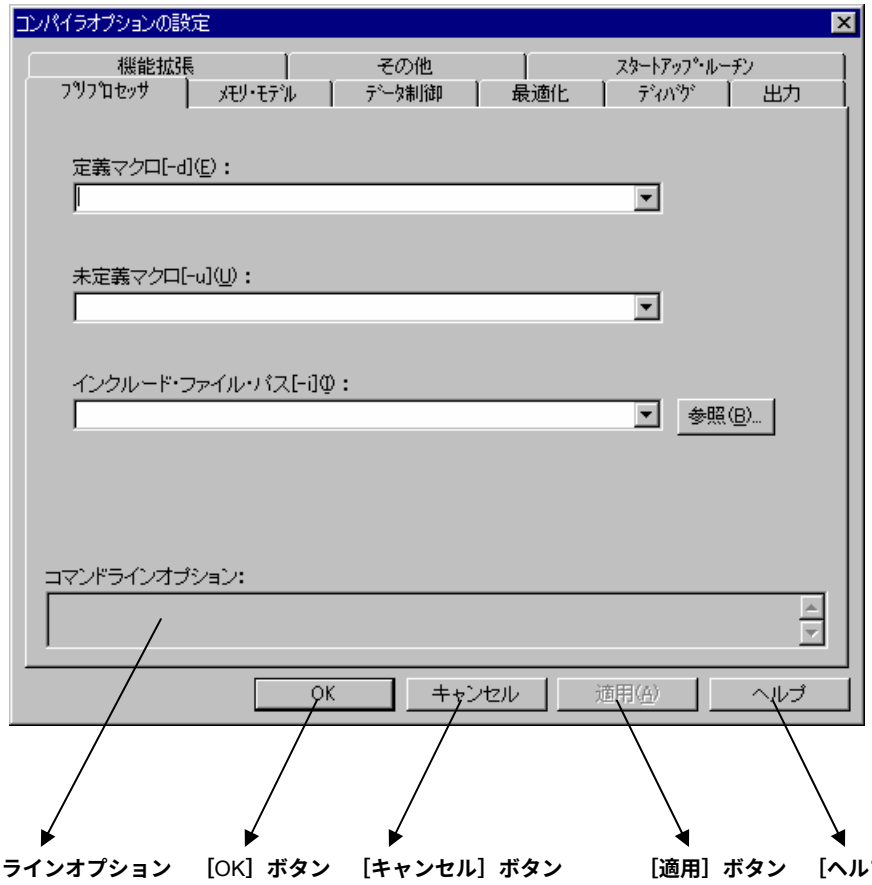
また，このダイアログは次の通りです。

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）

3. 1. 4　オプション設定ダイアログの各部の説明

ここでは，オプション設定ダイアログの各部について説明します。



- ［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でソース・ファイルが選択されている場合はそのファイルに対して，選択されていない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。

・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（T）」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。本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，  ボタンでスクロールすることにより確認できます。

・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（ディフォールト）

メモリ・モデル

データ制御

最適化

ディバグ

出力

機能拡張

その他

スタートアップ・ルーチン

(1) プリプロセッサを選択した場合の設定画面

・定義マクロ

-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。

・未定義マクロ

-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。

・インクルード・ファイル・パス

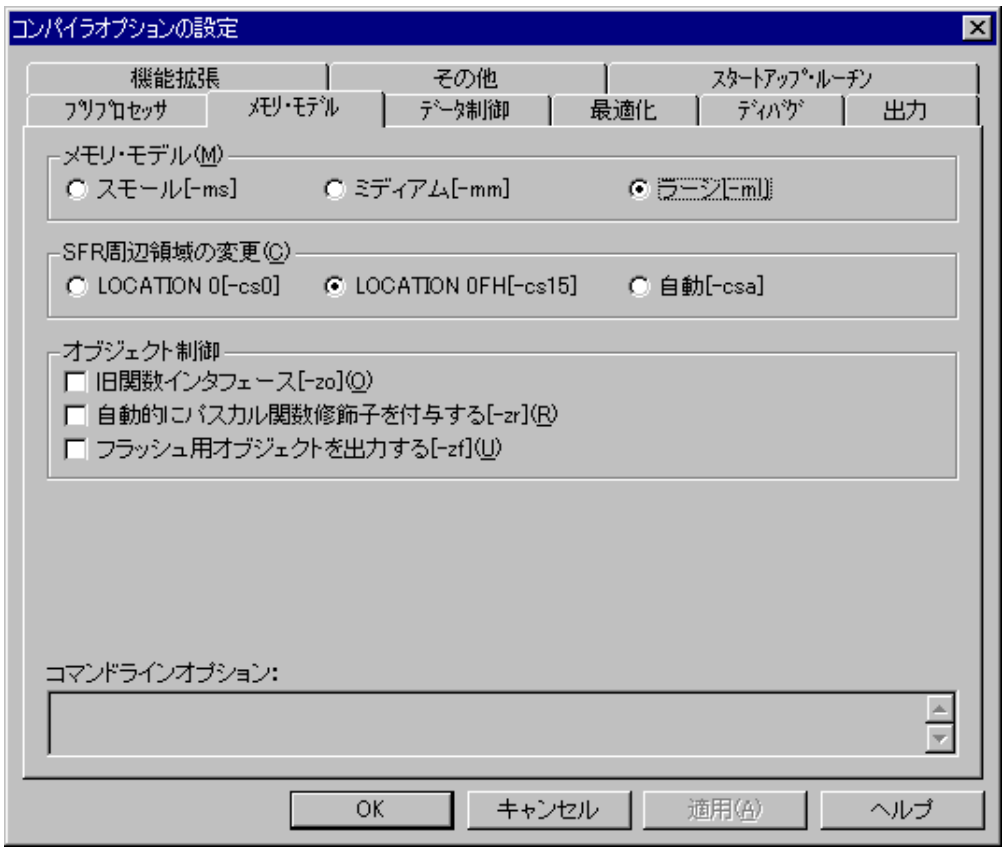
-Iで指定するインクルード・ファイルがあるディレクトリを，コンボ・ボックスに入力します。

‘ , ’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。

[参照（B）...] 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・モデルを選択した場合の設定画面



・メモリ・モデル

使用するメモリ・モデル・オプション-MS/-MM/-ML(**スモール／ミディアム／ラージ**)**をラジオ・ボタンで選択します。**

・SFR**周辺領域の変更**

使用するロケーション・オプション-CS0/-CS15/-CSA(LOCATION0/0FH/0AH)**をラジオ・ボタンで選択し，**saddr**領域の配置をコンパイラに指示します。**

・オブジェクト制御

旧関数インタフェース

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

自動的にパスカル関数修飾子を付与する

-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**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) データ制御を選択した場合の設定画面

- **外部変数をsaddr領域に割り当てる**

  -RD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。

  saddr領域に配置させる外部変数の種類をラジオ・ボタンをチェックすることにより選択します。

- **static変数をsaddr領域に割り当てる**

  -RS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。

  saddr領域に配置させるstatic変数の種類をラジオ・ボタンをチェックすることにより選択します。

- **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。

- **2バイト・アライメント**

  外部変数（SREGを除く）を2バイトアライメントする(ラージ・モデルのみ)

  -RAオプションを有効にする場合にチェック・ボックスをチェックします。

  構造体メンバを2バイトアライメントする

  -RP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) 最適化を選択した場合の設定画面

(a) 「グループ（G）」で「推奨統合オプション」を選択した場合

・推奨統合オプション

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，ディバグ制限の4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。

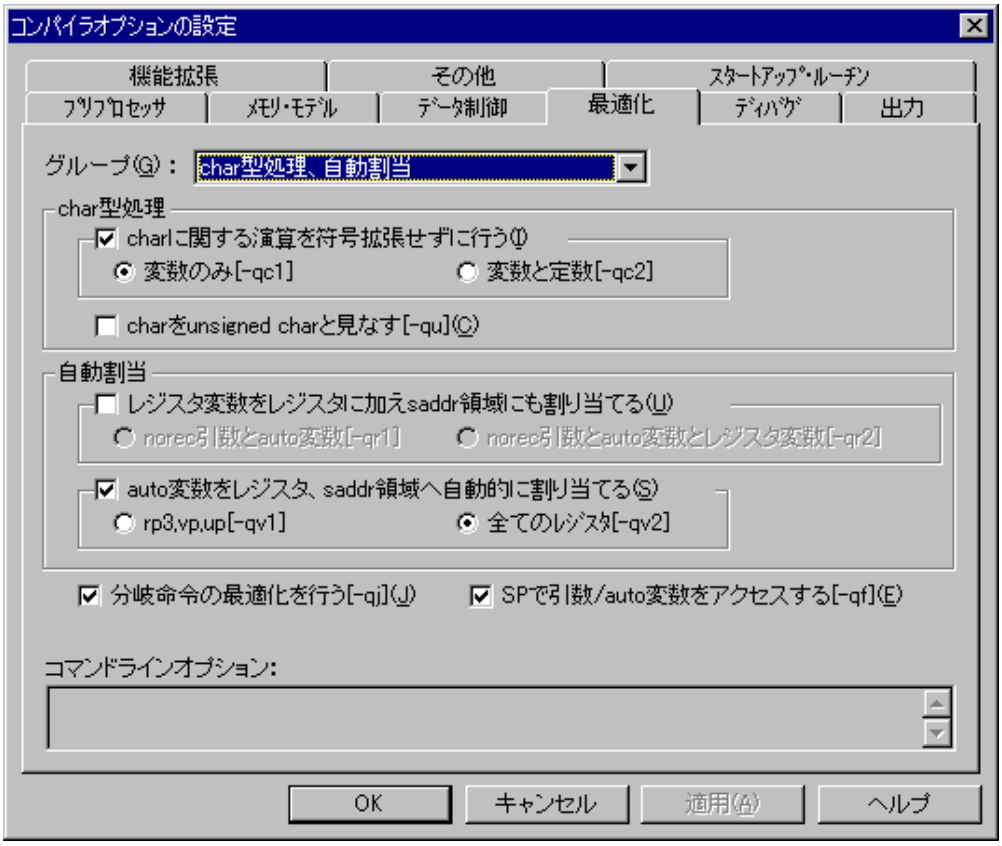
ディフォールト：-QX2オプションで，実行スピード，サイズ効率の両方とも重視する場合に選択します。

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。

ディバグ制限　：-QX4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，-QX3指定時よりもコード・サイズを減らしたい場合に使用しますが，コードの最適化箇所で一部ディバグ制限がある場合があります。

-QX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，上記の各パターンをラジオ・ボタンで選択します。

(b) 「グループ（G）」で「char型処理，自動割当」を選択した場合



・char型処理

charに関する演算を符号拡張せずに行う

-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，割り当てる変数をラジオ・ボタンで選択します。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

-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。

・分岐命令の最適化を行う

-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。

・SPで引数/auto変数をアクセスする

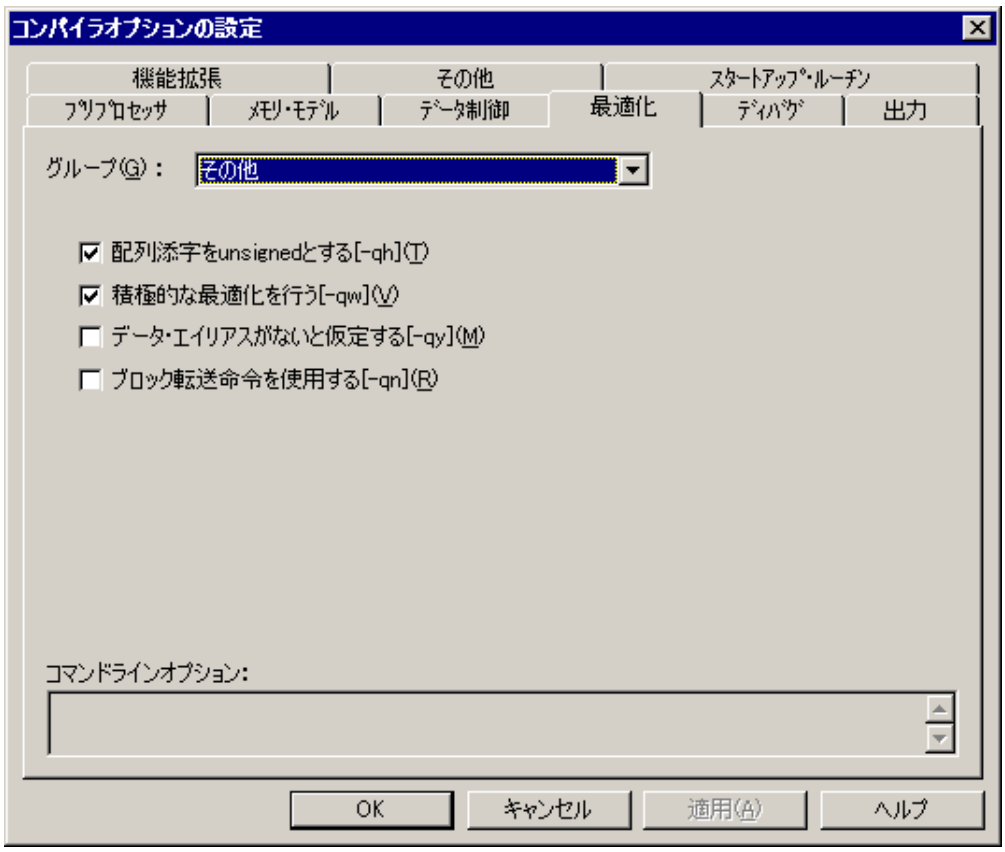
-QFオプションを有効にする場合にチェック・ボックスをチェックします。

(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

・ライブラリ呼び出し（サイズ優先最適化）

-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



- **配列添字をunsignedとする**

  -QH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。

- **積極的な最適化を行う**

  -QWオプションを有効にする場合にチェック・ボックスをチェックします。

- **データ・エイリアスがないと仮定する**

  -QY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。

- **ブロック転送命令を使用する**

  -QN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。

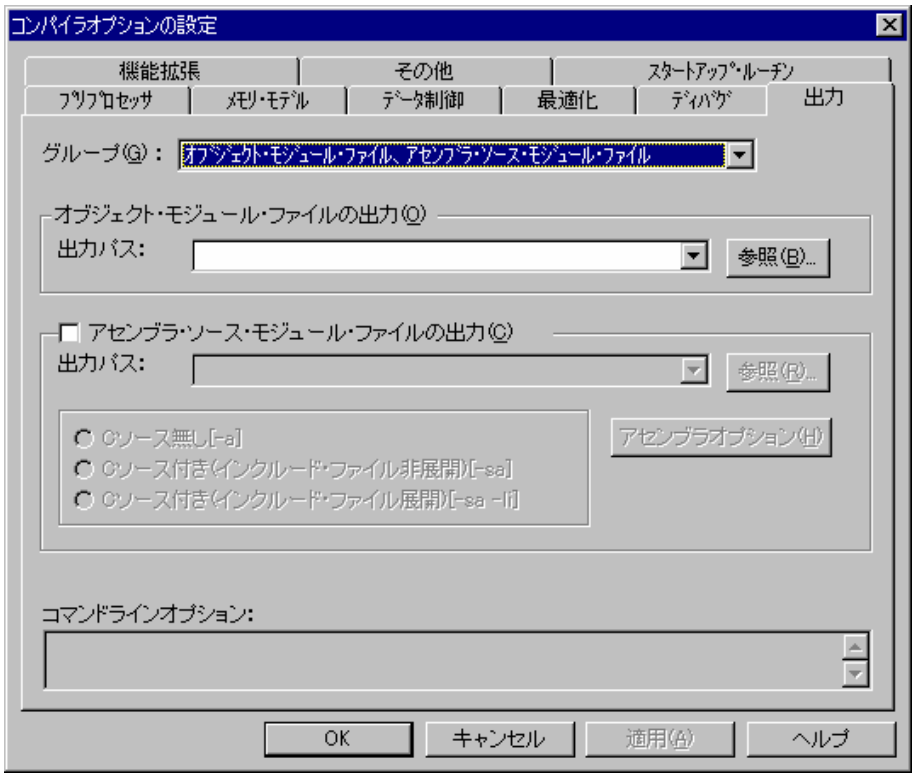
(5) ディバグを選択した場合の設定画面

・ディバグ情報の出力

-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) 出力を選択した場合の設定画面

(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



・オブジェクト・モジュール・ファイルの出力

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

-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（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。

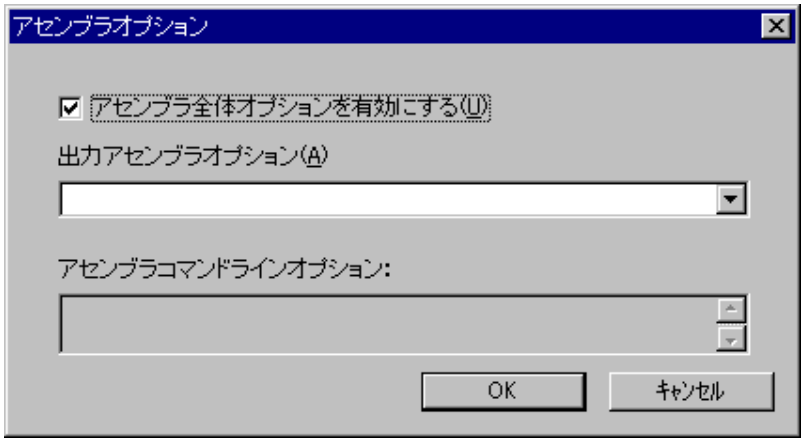
・［アセンブラオプション］ボタン

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。

・アセンブラオプションダイアログ

＜コンパイラオプションの設定＞ダイアログの出力タブの［アセンブラオプション］ボタンを押すと，次のダイアログが表示されます。



・アセンブラ全体オプションを有効にする

＜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

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 ▼ ボタンをクリックすることにより，過去に入力した履歴を選択できます。

注意　チップ種別指定（-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，本ツール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。

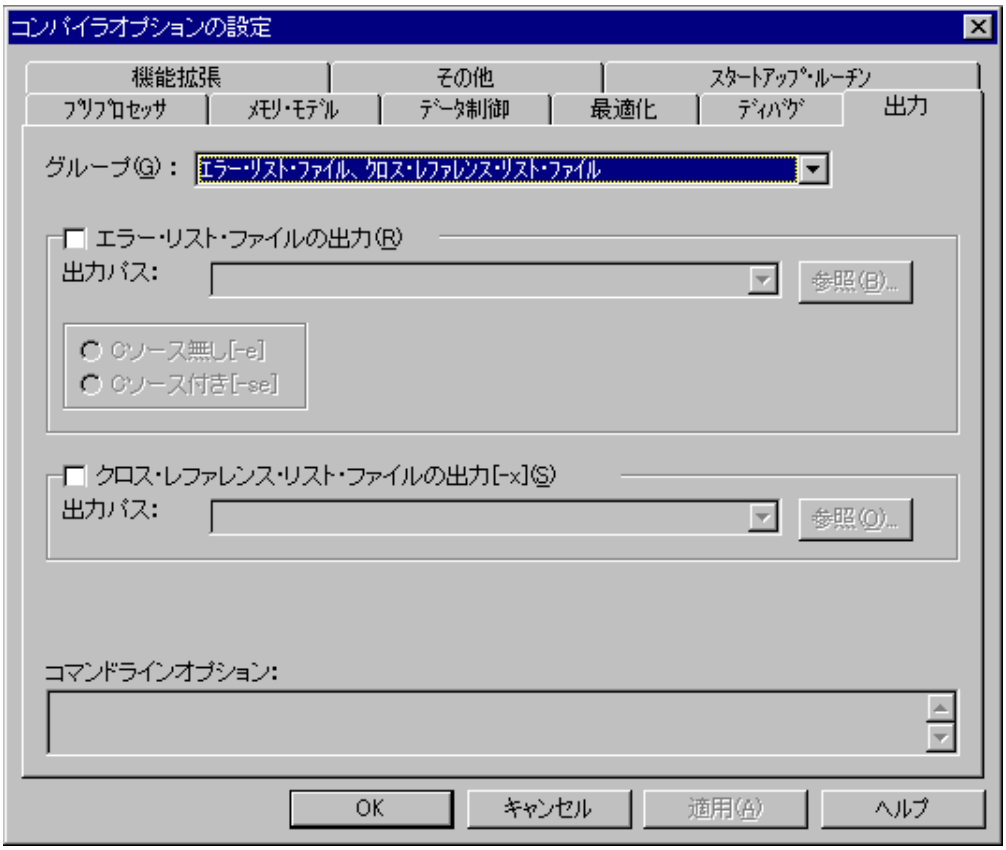
現在設定されているオプション文字列が表示されます。

文字列がボックス内に収まりきらない場合， ▲▼ ボタン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



・エラー・リスト・ファイルの出力

-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

・プリプロセス・リスト・ファイルの出力

-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。

・各種リストの設定

各種リスト出力オプションが設定されている場合に次のフォーマットの指定で出力されます。

1行文字数

-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，  ボタンで変更できます。

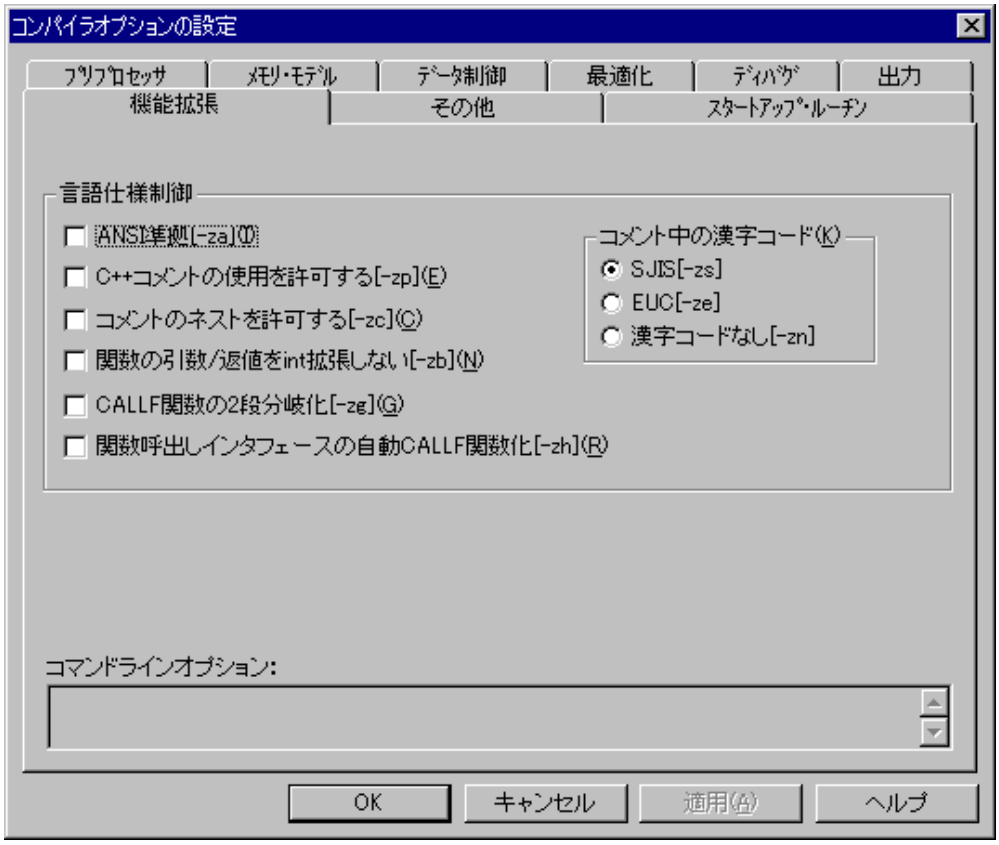
1ページ行数

-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，  ボタンで変更できます。

タブ文字長

-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，  ボタンで変更できます。

(7) 機能拡張を選択した場合の設定画面



・**言語仕様制御**

ANSI**準拠**

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

C++**コメントの使用を許可する**

-ZP**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**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**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。**

CALLF**関数の2段分岐化**

-ZG**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。**

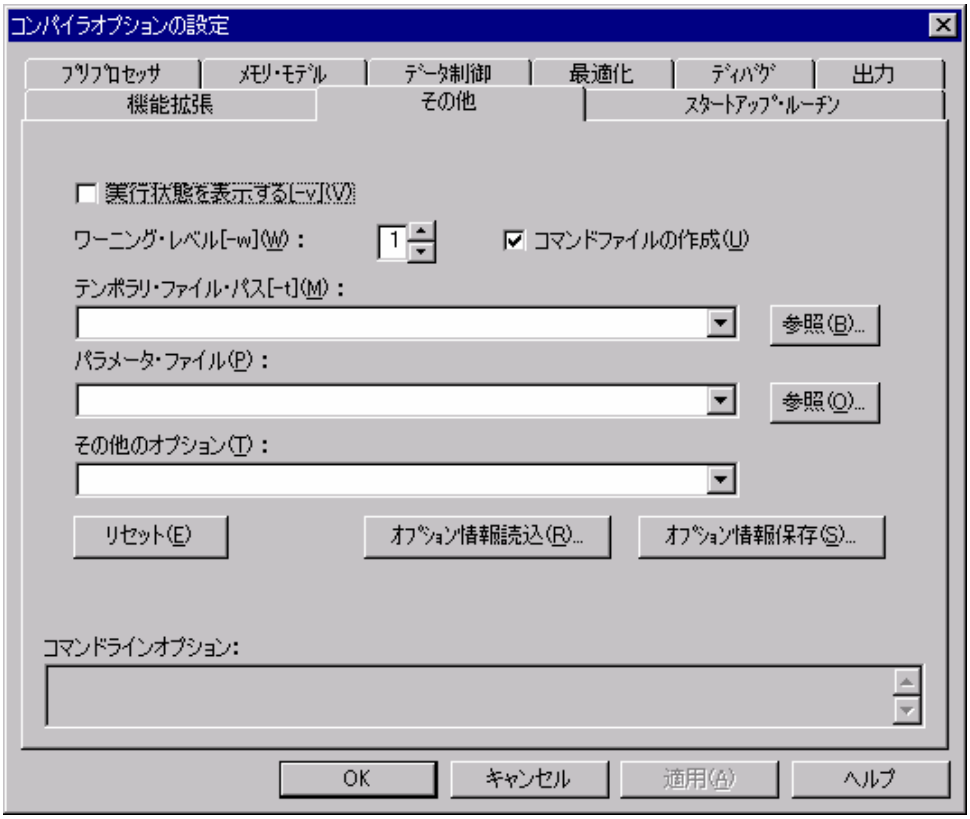
関数呼出しインタフェースの自動CALLF**関数化**

-ZH**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

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) その他を選択した場合の設定画面



・実行状態を表示する

-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オプションのレベルを変更する場合は，  ボタンで変更します。

・テンポラリ・ファイル・パス

-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

-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  ボタンをクリックすることにより，過去に入力した履歴を選択できます。

・その他のオプション

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  ボタンをクリックすることにより，過去に入力した履歴を選択できます。

・［リセット］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］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］ボタン

このボタンは，［OK］ボタンまたは［適用］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。

・コマンド・ファイルの作成

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。

(9) スタートアップ・ルーチンを選択した場合の設定画面

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

本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

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

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

本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。

「旧関数インタフェース」，「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。

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　コンパイルからリンクの手順（フラッシュ・メモリのセルフ書き換えモード未使用時）

3. 2. 1　PM plusからのMAKE

PC-9800シリーズ，IBM PC/AT互換機で，PM plusを使用してMAKE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。

3. 2. 2　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。

3. 2. 3　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“ワークスベース”を作成する必要があります。ワークスベースの作成方法はPM plus Ver.5.10　ユーザーズ・マニュアル（U16569J）を参照してください。

3. 2. 4　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［-QCFHJLVW］から，サイズ優先　［-QX3］に変更した例を示します。

図3－1　最適化オプションの選択

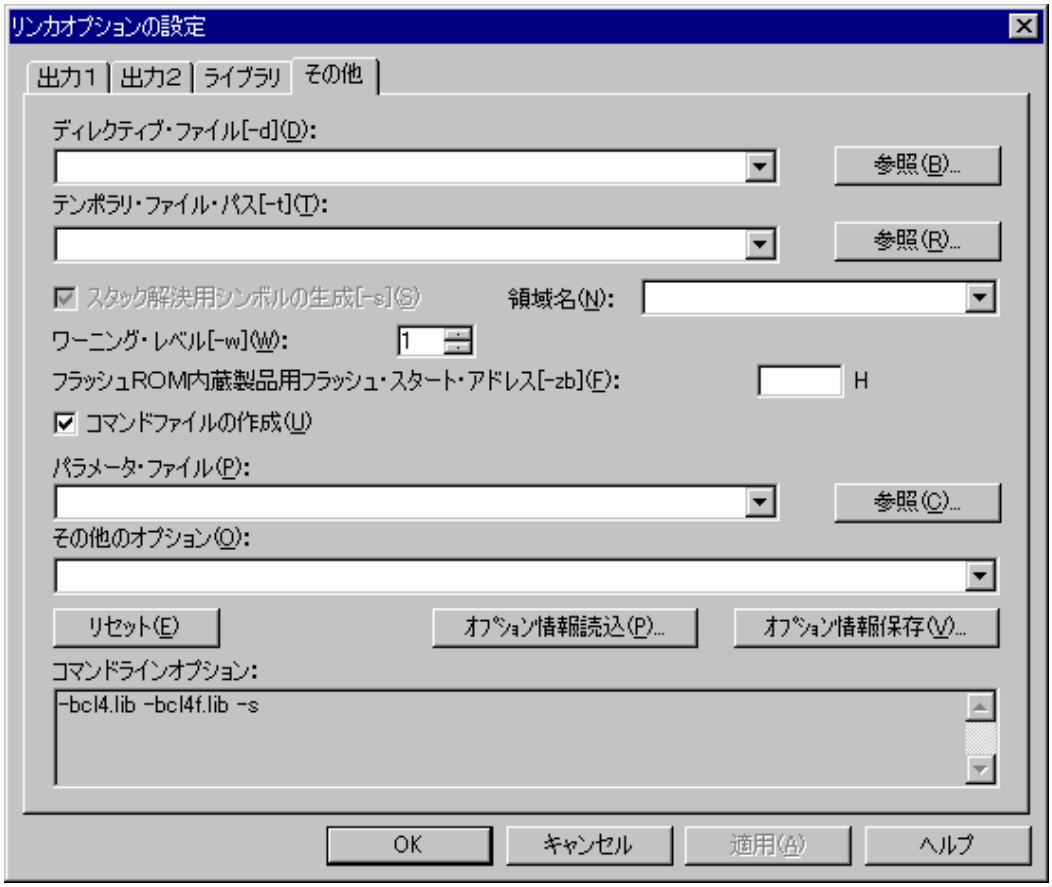
＜コンパイラオプションの設定＞ダイアログの《スタートアップ・ルーチン》で，標準のスタートアップ・ルーチンを使用する設定にしている場合は，本コンパイラ標準のスタートアップ・ルーチンが，すべてのソースの前でリンクされます（＜リンカオプションの設定＞に表示はされません）。

標準のライブラリを使用する設定にしている場合には，本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3－2　＜リンカオプションの設定＞ダイアログ



3. 2. 5　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  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。

3.2.6　コマンド行（DOSプロンプト，EWS）でのコンパイル～リンク

（1）パラメータ・ファイル未使用時

コマンド行で本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］CC78K4［△オプション］△ Cソース名［△オプション］
>［パス名］RA78K4［△オプション］△ アセンブラ・ソース名［△オプション］
>［パス名］LK78K4　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**

1. **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）**
2. **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）**
3. **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**

1. **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）**
2. **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）**
3. **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）

1. **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
2. **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>cc78k4 –c4038 s1.c –e -a -iC:¥nectools32¥inc78k4 –yC:¥nectools32¥dev
C>ra78k4 –c4038 s2.asm -e -yC:¥nectools32¥dev
C>lk78k4 s41.rel sl.rel s2.rel -bC:¥nectools32¥lib78k4¥cl4.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RA78K4　**アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編**（U16708J）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。

注意　パラメータ・ファイルは，PM plus**のオプション設定では指定できません。**

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］CC78K4 △ -Fパラメータ・ファイル名
>［パス名］RA78K4 △ -Fパラメータ・ファイル名
>［パス名］LK78K4 △ -Fパラメータ・ファイル名
```

次に使用例を示します。

```
例  C> cc78k4     -Fpara.pcc
    C> ra78k4     -Fpara.pra
    C> lk78k4     -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）

```
-c4038 sl.c -e -a -iC:¥nectools32¥inc78k4 -yC:¥nectools32¥dev
```

（para.praの内容）

```
-c4038 s2.asm -e -yC:¥nectools32¥dev
```

（para.plkの内容）

```
s41.rel s1.rel s2.rel -bC:¥nectools32¥lib78k4¥cl4.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RA78K4　**アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編**（U16708J）を参照してください。

3. 3　コンパイルからリンクの手順（フラッシュ・メモリの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～リンク

PC-9800シリーズ，IBM PC/AT互換機で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～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

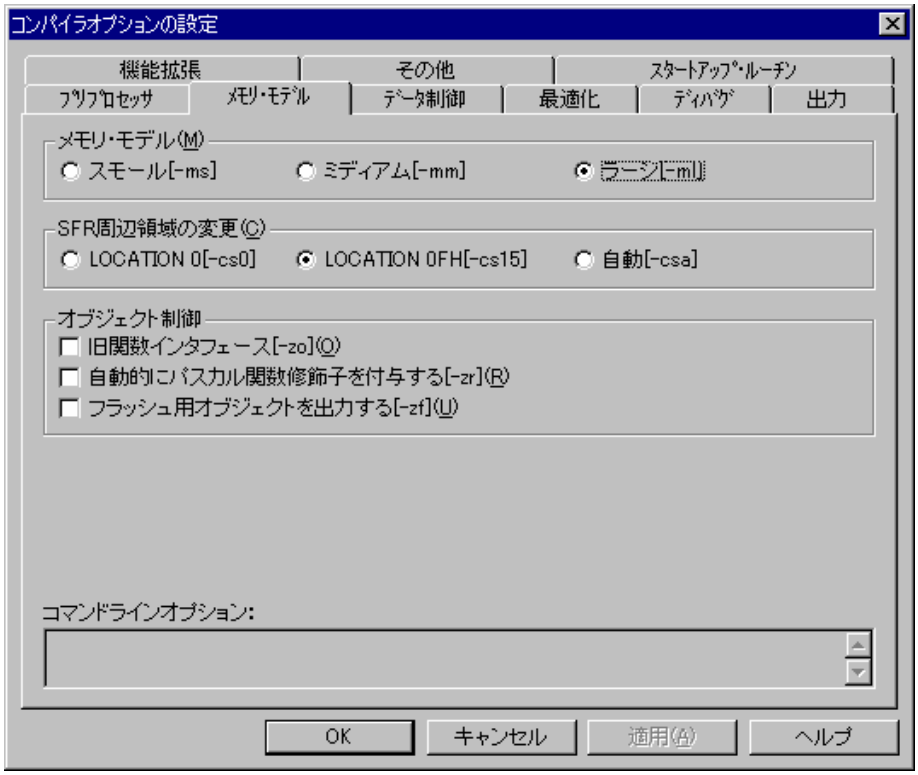
プロジェクト作成の終了時に自動的に作成されるメイク・ファイルは，ビルドのために必要最低限のオプションのみが設定されています。プロジェクト特有のオプションは，［ツール（T）］メニューで設定します。

［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。

① **コンパイラ・オプションの設定**

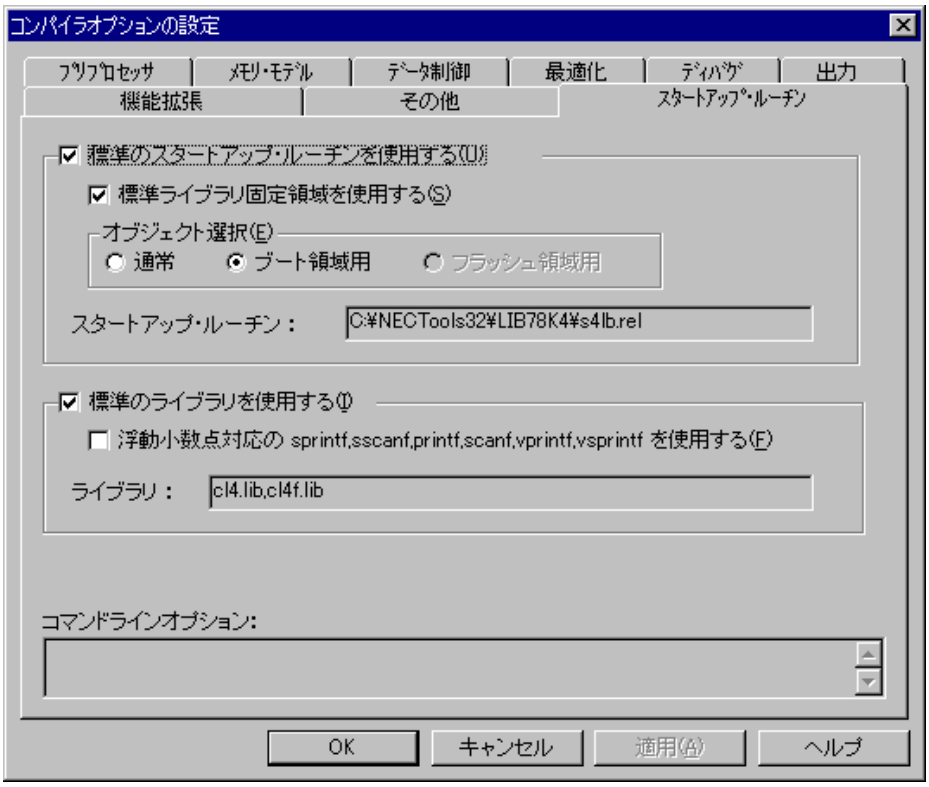
《メモリ・モデル》で，-ZFオプションは指定しないでください。

図3－3　<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》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3－4　ブート領域用の選択



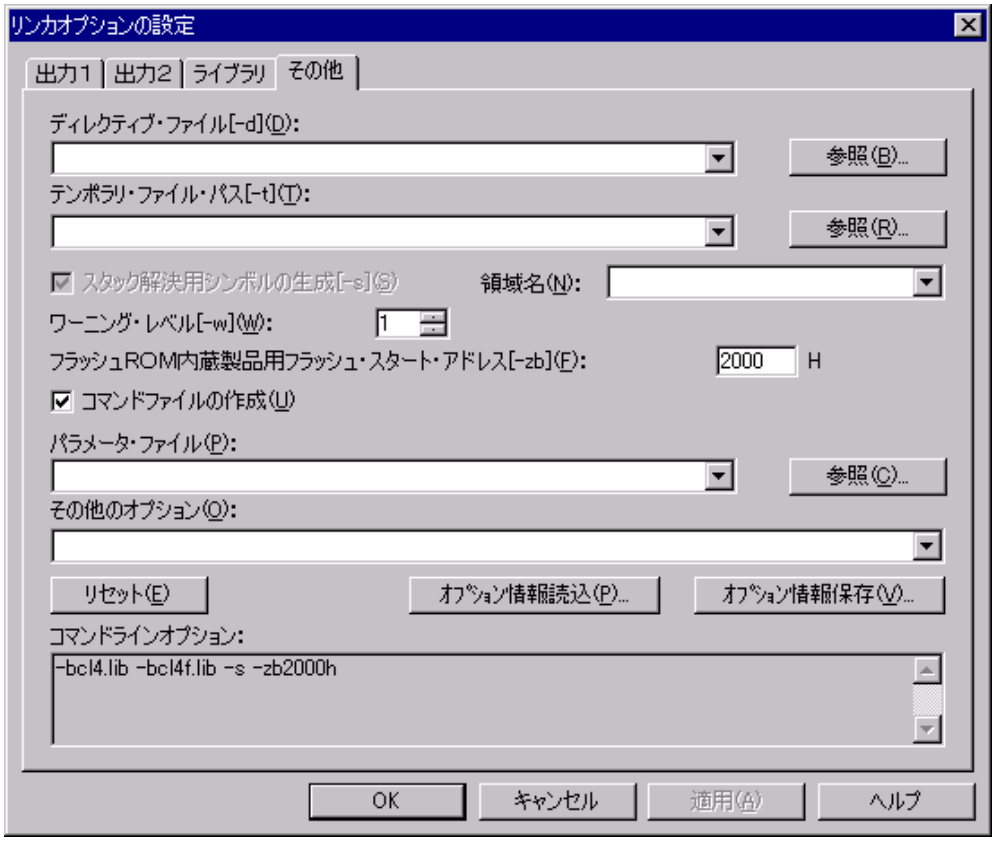
② **リンカ・オプションの設定**

フラッシュ・スタート・アドレス指定オプション-ZB**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》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。

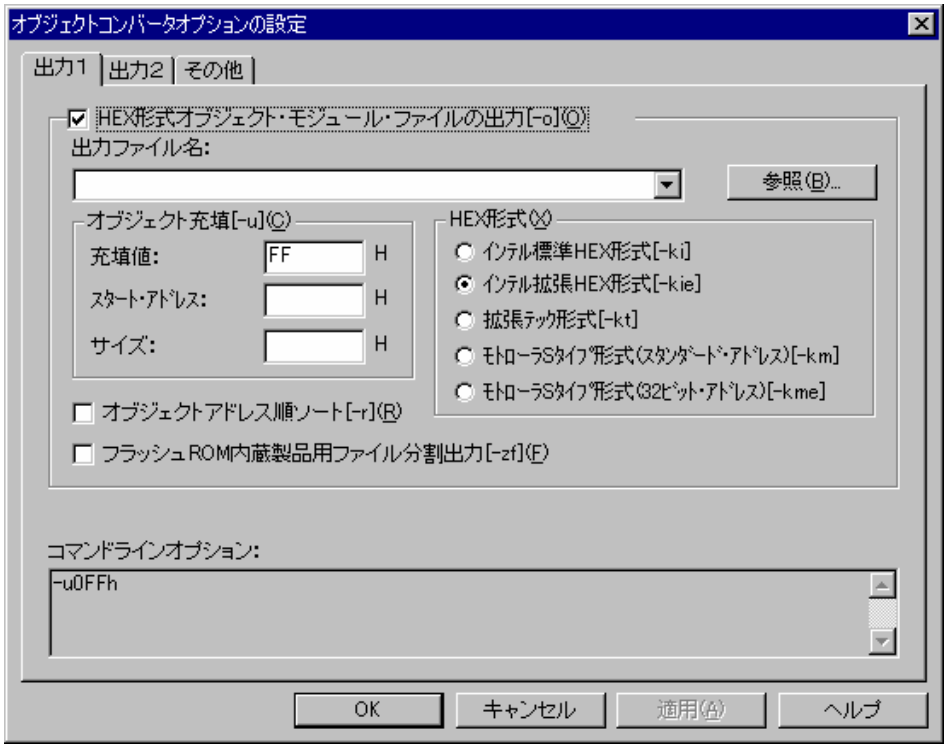
また，ソースファイル指定に，Cソース（boot.c）**が含まれているので，スタック・シンボル自動生成オプション**-S**が自動的に設定されます。**

備考　**リンカ・オプションについては，**RA78K4　**アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編（**U16708J**）を参照してください。**



③　**オブジェクト・コンバータ・オプションの設定**

オブジェクト・コンバータ・オプション-ZFを指定しないでください。



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

(c) プロジェクトのビルド

設定したオプションで，プロジェクトをビルドします。

プロジェクト全体のビルドは，［ビルド（B）］→［ビルド（B）］メニューから，またはツール・バーの  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

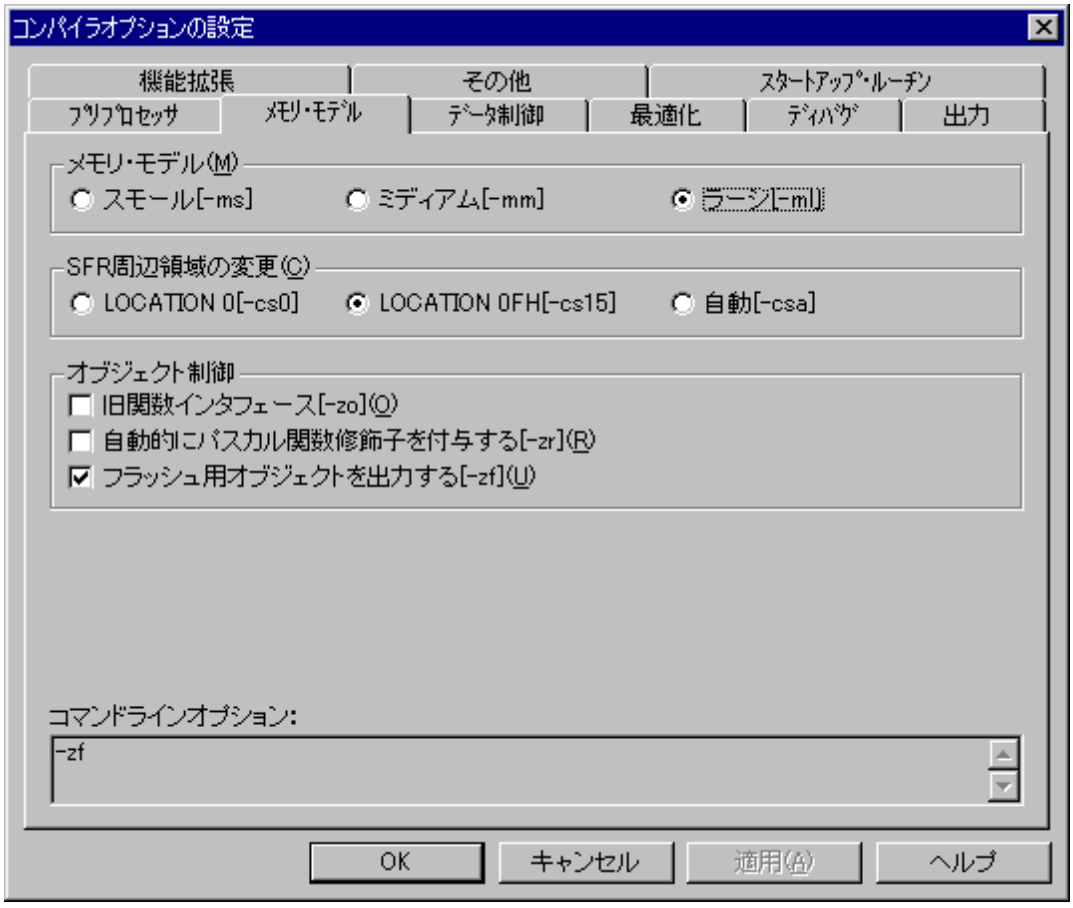
プロジェクト作成の終了時に自動的に作成されるメイク・ファイルは，ビルドのために必要最低限のオプションのみが設定されています。プロジェクト特有のオプションは，［ツール（T）］メニューで設定します。

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

① コンパイラ・オプションの設定

《メモリ・モデル》で，-ZFオプションを指定してください。

図3－5　＜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》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。

② リンカ・オプションの設定

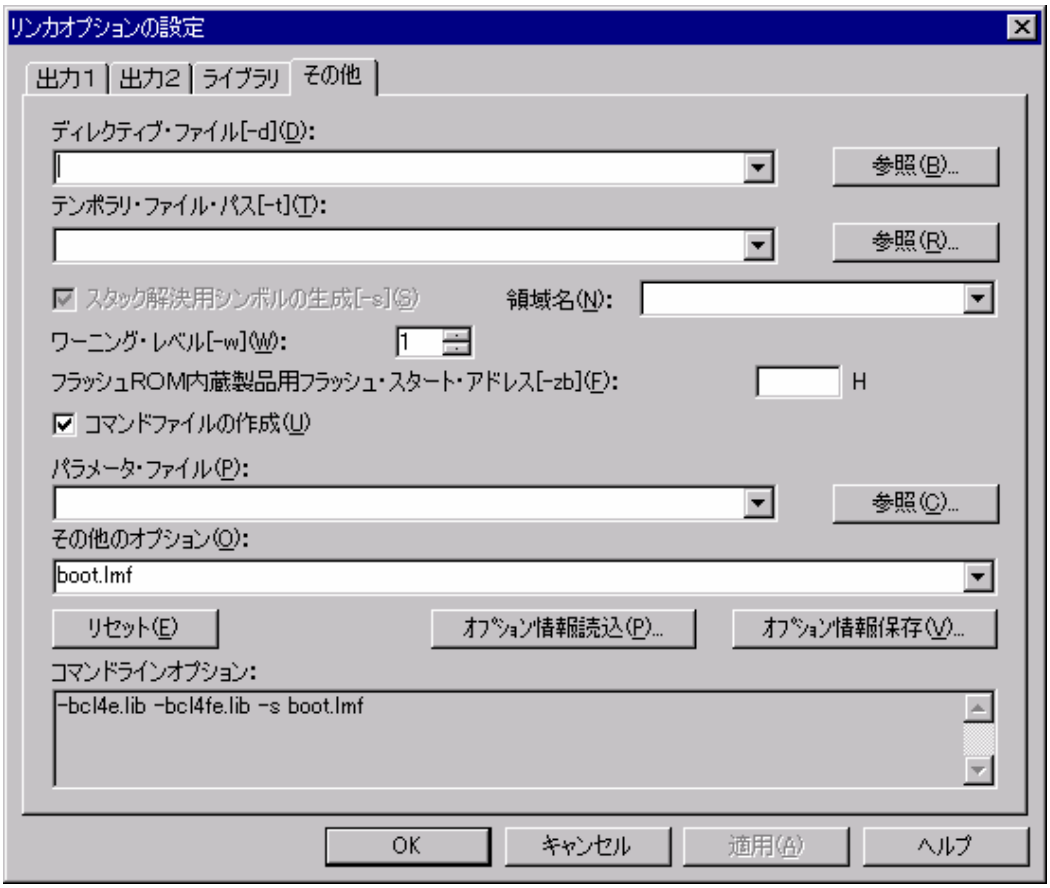
リンクするブート領域用ロード・モジュール・ファイルを指定し，［OK］ボタンを押します。

コンパイラ・オプション設定の《スタートアップ・ルーチン》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・シンボル自動生成オプション-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，RA78K4　**アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編**（U16708J）を参照してください。

図3－6　＜リンカオプションの設定＞ダイアログ



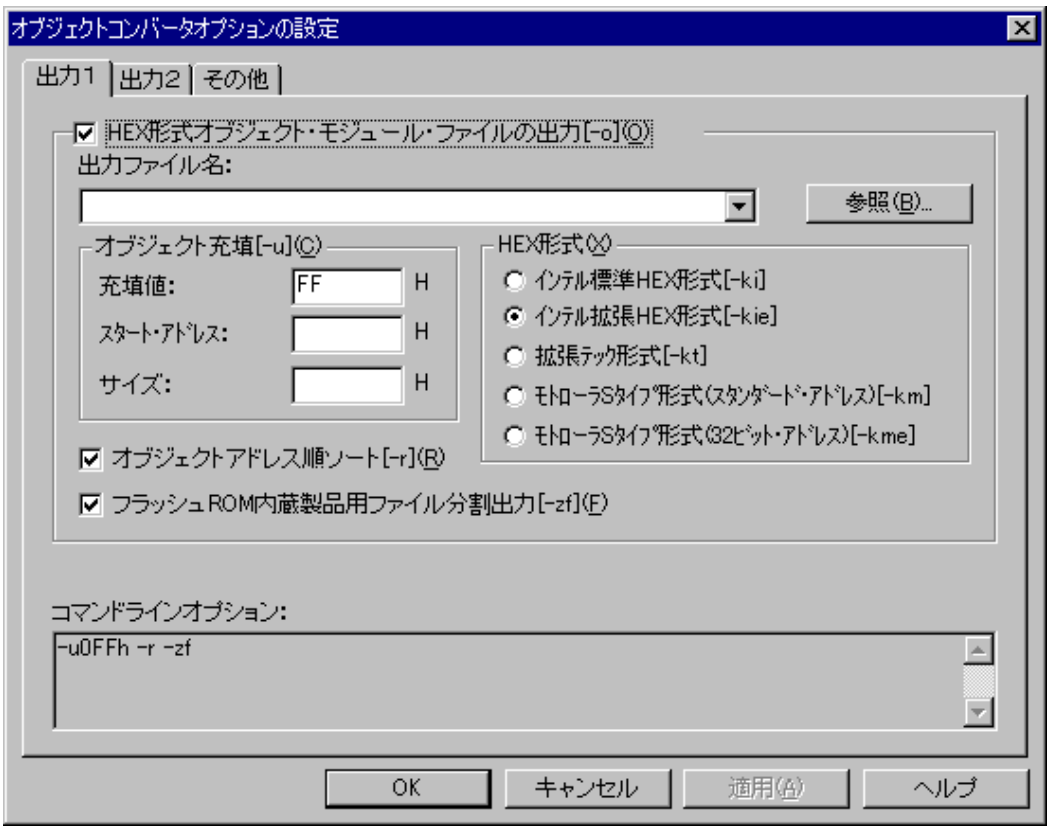
③　オブジェクト・コンバータ・オプションの設定（フラッシュ領域用）

オブジェクト・コンバータ・オプション-ZF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3－7　＜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  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，EWS）でのコンパイル～リンク

（1）パラメータ・ファイル未使用時

コマンド行で本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］CC78K4［△オプション］△Cソース名［△オプション］
>［パス名］RA78K4［△オプション］△アセンブラ・ソース名［△オプション］
>［パス名］LK78K4［△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～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> cc78k4 –cf4943 boot.c -iC:¥nectools32¥inc78k4 -yC:¥nectools32¥dev
```

②　ブート領域用プログラムのリンク

```
C> lk78k4 s41b.rel boot.rel -bC:¥nectools32¥lib78k4¥cl4.lib -s -oboot.lmf
   -zb2000h –yC:¥nectools32¥dev
```

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> oc78k4 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～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> cc78k4 –cf4943 flash.c -zf -iC:¥nectools32¥inc78k4
   -yC:¥nectools32¥dev
```

⑤　フラッシュ領域用プログラムのリンク

```
C> lk78k4 boot.lmf s4le.rel flash.rel -bC:¥nectools32¥lib78k4¥cl4e.lib
   -s –oflash.lmf -yC:¥nectools32¥dev
```

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> oc78k4 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**RA78K4　**アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編（**U16708J**）**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**

1. **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）**
2. 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）**
3. 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**

1. **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）**
2. 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）**
3. 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）

1. **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**
2. 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**
3. **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。

注意　パラメータ・ファイルは，PM plus**でのオプション設定では指定できません。**

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］CC78K4△-Fパラメータ・ファイル名
>［パス名］RA78K4△-Fパラメータ・ファイル名
>［パス名］LK78K4△-Fパラメータ・ファイル名
```

次に使用例を示します。

```
例  C>cc78k4 -Fpara.pcc
    C>lk78k4 -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)

```
-cf4943 boot.c -iC:¥nectools32¥inc78k4 -yC:¥nectools32¥dev
```

(para.plkの内容)

```
s4lb.rel boot.rel -bC:¥nectools32¥lib78k4¥cl4.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RA78K4　**アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編**（U16708J）を参照してください。

3.4　Cコンパイラの入出力ファイル

CC78K4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。

CC78K4の入出力ファイルを次に示します。

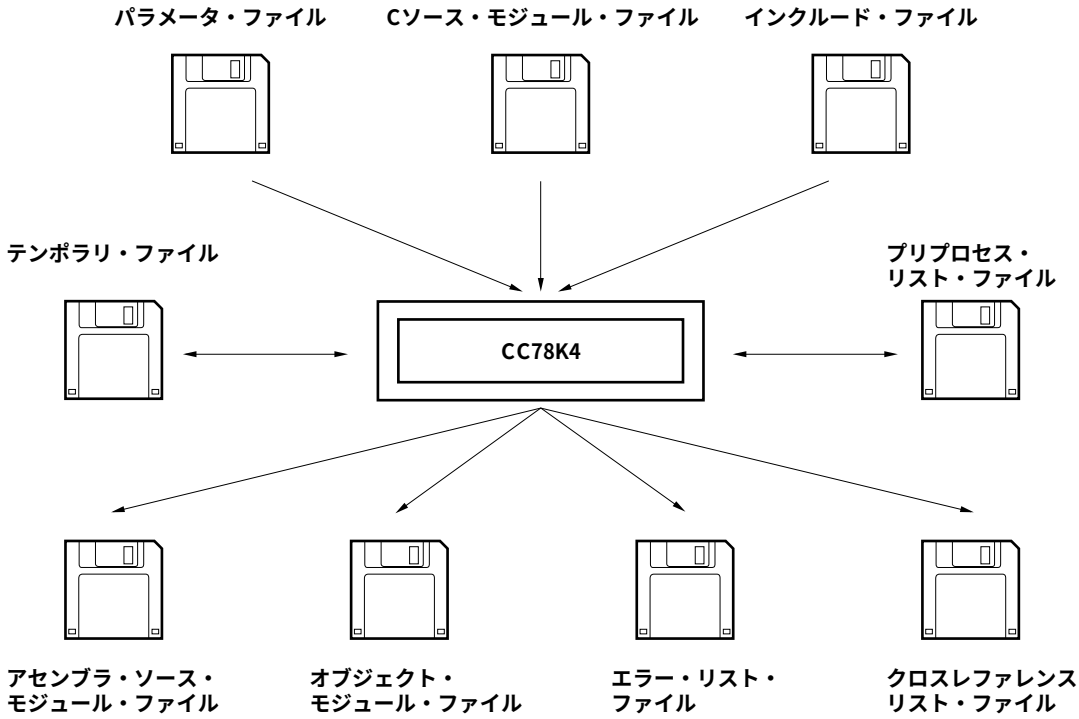
表3－1　Cコンパイラの入出力ファイル

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
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3－8　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。

3.5　実行開始・終了メッセージ

(1) 実行開始メッセージ

CC78K4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/IV Series C Compiler Vx.xx  [xx xxx xxxx]
    Copyright (C) NEC Electronics Corporation xxxx,xxxx
```

(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 : uPD784xxx
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) : W745 Expected function prototype
PRIME.C(20) : W745 Expected function prototype
PRIME.C(26) : W622 No return value
PRIME.C(37) : W622 No return value
PRIME.C(44) : W622 No return value

Target chip : uPD784xxx
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>cc78k4 –c4038 -e prime.c –s

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

A018 Option is not recognized ‘-s’
Please enter ‘CC78K4  -- ‘ , 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　CC78K4の機能

4.1　最適化手法

CC78K4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　最適化手法で示します。

表4－1　最適化手法（1/2）

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。
オプティマイザ	⑧	共通部分式の削除	a = b＋c； d = b＋c＋e；　→　a = b＋c； d = a＋e；

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

	⑩	無用命令の削除	a = a；　→ 削除 a = b；のあと，aが参照されません。　→ 削除 （aはオートマティック変数）
	⑪	複写の削除	a = b； c = a＋d；　→　c = b＋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

表4－1　最適化手法（2/2）

フェーズ		内　容	例
オプティマイザ	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/-RS/-RD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。

CC78K4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）s4*.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）cl4*.lib：CC78K4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。

① ランタイム・ライブラリ

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

② 標準ライブラリ

標準ライブラリ名はシンボルの先頭に_が付加されます。

（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　CC78K4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

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　コンパイラ・オプションの指定方法

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・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　（△：スペースなどの空白）

```
CC78K4△-c4038△prime.c△-aprime.asm△-qx3
```

5. 2　コンパイラ・オプションの優先度

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										×	△

←横軸

↑
縦軸

【×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>cc78k4  -c4038  -e  sample.c  -no  -rd  -g
```

-RD, -Gオプションは，無効となります。

```
例2.  C>cc78k4  -c4038  -e  sample.c  -p  -k
```

-Pオプションが指定されているので-Kオプションは有効です。

```
例3.  C>cc78k4  -c4038  -e  sample.c  -utest  -dtest=1
```

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

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

```
例4.  C>cc78k4  -c4038  -e  sample.c  -o  -no
```

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

表5－1　コンパイラ・オプションの優先度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“ -- ”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　コンパイラ・オプションの説明

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CC78K4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>cc78k4  -c4038  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（-C）

-C	デバイス種別指定

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。

・CC78K4使用時には，デバイス・ファイルが必要となります。デバイス・ファイルは，DEVディレクトリか，または，BINディレクトリにコピーして使用してください。

【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。

-C	デバイス種別指定

【使 用 例】

・コマンド行で指定します。ターゲット・デバイスはμPD784038**です。**

```
C>cc78k4  -c4038  prime.c
```

・Cソース中で指定して，起動させます。

```
#pragma         pc(4038)
#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>cc78k4  prime.c
```

-C　　デバイス種別指定

・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(4038)
#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>cc78k4  -c4026  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/IV Series C Compiler Vx.xx  [xx xxx xxxx]
   Copyright (C) NEC Electronics Corporation xxxx,xxxx

sample¥prime.c(1)  : W832 Duplicated chip specifier
sample¥prime.c(18) : W745 Expected function prototype
sample¥prime.c(20) : W745 Expected function prototype
sample¥prime.c(26) : W622 No return value
sample¥prime.c(37) : W622 No return value
sample¥prime.c(44) : W622 No return value

Target chip : uPD784026
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（-O/-NO）

-O/-NO	オブジェクト・モジュール・ファイル作成指定

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》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>cc78k4  -c4038  prime.c  -no  -o
```

（3）メモリ配置指定（-R/-NR, -RD/-NR, -RS/-NR, -RA/-NR, -RP/-NR, -RB/-NR）

-R/-NR　　**メモリ配置指定**

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，アボート・エラー（**A012**）となります。**

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］（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。**
S［n］（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。**
A	2**バイト以上の外部変数（**saddr**に割り付けられる変数を除く）を**2**バイト・アライメントして割り付けます。また構造体を**Non-packing**とします。**
P	**構造体を**Non-packing**とします。**

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。**

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。**
S	saddr**領域に自動的に割り付けません。**
A	2**バイト・アライメントしません。また，構造体を**packing**します。**
P	**構造体を**packing**します。**

【使 用 例】

```
C>cc78k4  -c4038  -rds
```

-RD/-NR　　**メモリ配置指定**

記述形式	-RD［n］（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**の値によって，割り付ける変数の最大幅を次のように指定します。**

n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, **（スモール・モデルおよびミディアム・モデルのデータへの）ポインタ**
4	char, unsigned char, short, unsigned short, int, unsigned int, enum, **（スモール・モデルおよびミディアム・モデルのデータへの）ポインタ，**long, unsigned long, **（ミディアム・モデルおよびラージ・モデルの関数への）ポインタ**
省略	**すべての変数（構造体，共用体を含む）**

- 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。**

-RS/-NR　　**メモリ配置指定**

記述形式	-RS［n］（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**の値によって，割り付ける変数の最大幅を次のように指定します。**

n**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, **（スモール・モデルおよびミディアム・モデルのデータへの）ポインタ**
4	char, unsigned char, short, unsigned short, int, unsigned int, enum, **（スモール・モデルおよびミディアム・モデルのデータへの）ポインタ，**long, unsigned long, **（ミディアム・モデルおよびラージ・モデルの関数への）ポインタ**
省略	**すべての変数（構造体，共用体を含む）**

- 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。**
- 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**宣言された**static auto**変数と同様に扱います。**

(4) 最適化指定（-Q/-NQ）

-Q/-NQ	最適化指定

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	-QCFHJLVW

【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5－2　最適化種別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5－2に示します。

表5－2　最適化種別（1/2）

最適化種別	処理内容
省略	-QCFHJLVW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。
X［n］ (n = 1-4)	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，-QCFHJVW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，-QCFHJL3VWを指定したものと見なす 4：コード・サイズ優先として，-QCFHJL4VWを指定したものと見なす（一部ディバグ制限があります）

-Q/-NQ　　最適化指定

表5－2　最適化種別（2/2）

最適化種別	処理内容
F	フレーム・ポインタ（スタック上にある関数引数や自動変数をアクセスするために使用するレジスタ）として，UUPレジスタではなくSPを使用し，空いたUUPレジスタをregister変数などに利用することにより，実行スピード，コード効率を向上させます。-QFオプション未指定時は，UUPレジスタをフレーム・ポインタとして使用します。
H	配列やポインタのオフセット計算を符号なしで行なうことにより，コード効率が向上します。このオプション使用時は，次の制限がありますのでご注意ください。 ・配列要素やポインタによるオブジェクトへのアクセスは64 Kバイト以内しかできません。 ・マイナス方向のオフセット計算はできません。
W	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。
V［n］ （n = 1, 2）	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。 Rサブオプションが同時に指定されている場合は，レジスタとsaddr領域に，そうでない場合はレジスタのみ割り付けます。 また，-ZOオプションが指定されていない場合は，パラメータにもオートマティック変数を割り付けます。nの値によって，自動割り当てす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：rp3, vp, upのみ割り当てる 2：すべてのレジスタに割り当てる
N	ブロック転送命令を使用したコード生成を行います。

Y	データ・エイリアスがないと仮定してコンパイルを行うため，コード効率が向上します。
	このオプション使用時には，同一関数内で，実体の直接アドレスとその実体を指すポインタによる間接アクセスを混在して記述した場合，不正コードになる可能性がありますので注意してください。
	（不正コードとなる例）
	`int i,j;`
	`int *p = &i;`
	`void main()`
	`{`
	`*p = 2;`
	`i = 1;`
	`j = *p;  /*-QYオプション指定時には，jに 2が代入され，未指定時には，1が代入される*/`
	`}`

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 ：配列へのアクセス・コード（ラージ・モデル時のみ有効） 3 ：2に加えて，四則演算，1命令単位 4 ：3に加えて，long型，unsigned long型のシンボルと定数またはレジスタの代入コード（実行時ルーチンからのリターン・アドレスが変化するため，ディバグ制限が生じます。）

注　CC78K4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。

-Q/-NQ	最適化指定

例　-QC2オプション指定時

```
int  i;

i = (int)20*20;                /* 400 */
```

- 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，-QCFHJLVW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**　–QFを指定　→ -QCHJLVW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。

【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>cc78k4  -c4038  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>cc78k4  -c4038  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>cc78k4  -c4038  prime.c  -qcr
```

(5) ディバグ情報出力指定 (-G/-NG)

-G/-NG	ディバグ情報出力指定

記述形式	-G［n］（n = 1, 2）
	-NG
省略時解釈	-G2

【機　　能】

- -Gオプションは，オブジェクト・モジュール・ファイル中にデ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。

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>cc78k4  -c4038  prime.c  -g
```

(6) プリプロセス・リスト・ファイル作成指定（-P, -K）

-P	プリプロセス・リスト・ファイル作成指定

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》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>cc78k4  -c4038  prime.c  -psample.ppl
```

-K　プリプロセス・リスト・ファイル作成指定

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5－3　-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。

-K	プリプロセス・リスト・ファイル作成指定

【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>cc78k4  -c4038  prime.c  -p  -kcn
```

prime.ppl**を参照します。**

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

Command   : -c4038 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 : uPD784038
Device file : VX.XX
*/
```

（7）プリプロセス指定（-D, -U, -I）

-D	プリプロセス指定

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>cc78k4  -c4038  prime.c  -dTEST,TIME=10
```

-U　　プリプロセス指定

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>cc78k4  -c4038  prime.c  -dTEST  -uTEST
```

-I	プリプロセス指定

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	・ソース・ファイルのあるディレクトリ[注1] ・環境変数INC78K4により指定されたディレクトリ ・C：¥NECTools32¥INC78K4[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。
　・ソース・ファイルのあるディレクトリ[注1]
　・-Iオプションにより指定されたディレクトリ
　・環境変数INC78K4により指定されたディレクトリ
　・C：¥NECTools32¥INC78K4[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。
2. C：¥NECTools32にインストールした場合の例です（Windows版のみ）。

【使 用 例】

・-Iオプションを指定します。

```
C>cc78k4  -c4038  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（-A, -SA）

-A	アセンブラ・ソース・モジュール・ファイル作成指定

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》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》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>cc78k4  -c4038  prime.c  -asample.asm
```

・アセンブラ・ソース・モジュール・ファイルをプリンタに出力します。

```
C>cc78k4  -c4038  prime.c -aprn
```

-SA　　**アセンブラ・ソース・モジュール・ファイル作成指定**

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》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。**

-SA	アセンブラ・ソース・モジュール・ファイル作成指定

【使 用 例】

・-SA**オプションを指定します。**

```
        C>cc78k4  -c4038  prime.c  -sa
```

prime.asm**を参照します。**

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

; Command   : -c4038 prime.c -sa
; In-file   : prime.c
; Asm-file  : prime.asm
; Para-file :

$CHGSFR(15)
$PROCESSOR(4038)
$DEBUG
$NODEBUGA
$KANJICODE SJIS
$TOL_INF       03FH, 0230H, 02H, 08021H, 00H

$DGS    FIL_NAM, .file,        03BH,   0FFFEH, 03FH,   067H,   01H,    00H
$DGS    AUX_FIL, prime.c
$DGS    MOD_NAM, prime,        00H,    0FFFEH, 00H,    077H,   00H,    00H

           :

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

           :

@@CODE  CSEG
_main:
$DGL    1,19
       push    uup
       push    rp3
       push    vvp
       push    ax
??bf_main:
; line    9 :  int i, prime, k, count;
; line   10 :
; line   11 :  count = 0;
$DGL    0,4
       subw    ax,ax
       movw    [sp+0],ax       ; count
; line   12 :
; line   13 :  for ( i = 0 ; i <= SIZE ; i++)
```

-SA　　アセンブラ・ソース・モジュール・ファイル作成指定

```
$DGL    0,6
       subw    rp3,rp3
?L0003:
       cmpw    rp3,#0C8H      ; 200
       bgt     $?L0004
; line    14 :         mark[i] = TRUE;
$DGL    0,7
       movw    de,rp3
       mov     a,#01H  ; 1
       mov     _mark[de],a
       incw    rp3
       br      $?L0003

           :

       END


; Target chip : uPD784038
; 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（-E, -SE）

-E	エラー・リスト・ファイル作成指定

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》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》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>cc78k4  -c4038  prime.c  -e
```

-E　　**エラー・リスト・ファイル作成指定**

エラー・リスト・ファイルを参照します。

```
prime.c(   18) : W745 Expected function prototype
prime.c(   20) : W745 Expected function prototype
prime.c(   26) : W622 No return value
prime.c(   37) : W622 No return value
prime.c(   44) : W622 No return value

Target chip : uPD784038
Device file : Vx.xx

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

-SE　　**エラー・リスト・ファイル作成指定**

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，‘ 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》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。

-SE	エラー・リスト・ファイル作成指定

【使 用 例】

・-SEオプションを指定します。

```
        C>cc78k4  -c4038  prime.c  -se
```

prime.cer**を参照します。**

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

Command         : -c4038  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);
*** WARNING W745 Expected function prototype
                        count++;
                                if((count%8) == 0) putchar('¥n');
*** WARNING W745 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（-X）

-X	クロスレファレンス・リスト・ファイル作成指定

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。
・致命的エラー（A024を除くアボート，F101）を除くコンパイル・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》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> cc78k4  -c4038  prime.c  -x
```

-X　　**クロスレファレンス・リスト・ファイル作成指定**

prime.xrf**を参照します。**

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

Command   : -c4038 prime -x
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
REG1          int     prime            9      17     18     21     21
REG1          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
PARAM         int     i               30      35
REG1          int     j               32      35
REG1          pointer ss              33      36
REG1          char    c               40      43
PARAM
REG1          char    d               42      43
              #define TRUE             1      14
              #define FALSE            2      22
              #define SIZE             3       5     13     15     21


 Target chip : uPD784038
 Device file : VX.XX
```

（11）リスト形式指定（-LW, -LL, -LT, -LF, -LI）

-LW	リスト形式指定

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> cc78k4  -c4038  prime.c  -x
```

-LL　　**リスト形式指定**

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，全角にしません。**

・**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> cc78k4  -c4038  prime.c  -x  -ll20
```

-LT　　**リスト形式指定**

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> cc78k4  -c4038  prime.c  -p
```

・HT**コードによるブランクを**1**に指定します。**

```
C> cc78k4  -c4038  prime.c  -p  -lt1
```

-LF　　**リスト形式指定**

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> cc78k4  -c4038  prime.c  -a  -lf
```

-LI　　**リスト形式指定**

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> cc78k4  -c4038  prime.c  -sa  -li
```

（12）ワーニング出力指定（-W）

-W	ワーニング出力指定

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かしないかを指定します。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5－4　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> cc78k4  -c4038  prime.c
```

（13）実行状態表示指定（-V/-NV）

-V/-NV	実行状態表示指定

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> cc78k4  -c4038  prime.c  -v
```

（14）パラメータ・ファイル指定（-F）

-F	パラメータ・ファイル指定

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。

【注　　意】

・PM plus使用時には，このオプションは使用できません（エラーとなります）。

【使 用 例】

・パラメータ・ファイルprime.pccの内容

```
; parameter file
prime.c  -c4038  -aprime.asm  -e  -x
```

prime.pccを使用してコンパイルします。

```
C> cc78k4  -fprime.pcc
```

（15）テンポラリ・ファイル作成ディレクトリ指定（-T）

-T	テンポラリ・ファイル作成ディレクトリ指定

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，Windowsベースの場合，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。UNIXベースの場合は，/tmp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> cc78k4  -c4038  prime.c  -ttmp
```

（16）ヘルプ指定（--/-?/-H）

--/-？/-H	ヘルプ指定

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> cc78k4  -H
```

（17）メモリ・モデル指定（-MS/-MM/-ML）

-MS/-MM/-ML	メモリ・モデル指定

記述形式	-MS
	-MM
	-ML
省略時解釈	-ML

【機　　能】

・次のメモリ・モデルでコンパイルすることを指定します。

-MS ：スモール・モデル

-MM ：ミディアム・モデル

-ML ：ラージ・モデル

【用　　途】

・プログラム領域，データ領域を変更できます。

【説　　明】

・スモール・モデルは，プログラム領域，データ領域あわせて64 Kバイトのモデルです。

・ミディアム・モデルは，プログラム領域1 Mバイト，データ領域64 Kバイト，あわせて1 Mバイトのモデルです。

・ラージ・モデルは，プログラム領域1 Mバイト，データ領域16 Mバイト，あわせて16 Mバイトのモデルです。

・前記すべてのデータ領域は，スタック領域を含みます。

【使 用 例】

・スモール・モデルを指定します。

```
C> cc78k4 -c4038 prime.c -ms
```

・ミディアム・モデルを指定します。

```
C> cc78k4 -c4038 prime.c -mm -cs15
```

（18）ロケーション機能指定（-CS）

-CS	ロケーション機能指定

記述形式	-CS［n］	（n：配置アドレス）
省略時解釈	スモール・モデルモデル	：saddr領域をFD20H-FFFFHに置く
	ミディアム・モデル	：saddr領域をFFD20H-FFFFFHに置く
	ラージ・モデル	：saddr領域をFFD20H-FFFFFHに置く

【機　　能】

- -CSオプションは，ロケーション機能を指定します。
- ロケーション機能は，78K4シリーズが持つ，内部データ領域（内部RAM領域とSFR領域）のアドレスを変更する機能です。

【用　　途】

- -CS0指定時は，内部データ領域としてLOCATION 0命令実行時のマッピングを指定します。
- -CS15，または -CS0FH指定時ではLOCATION OFH命令実行時のマッピングを指定します。
- -CSA指定時は，内部データ領域のマッピングに関連したチェックをリンカに任せます。

【説　　明】

- 配置アドレスを0, 15, Aのいずれか選択すると，saddr領域の配置を表5－5のように指定できます。
- スモール・モデルの場合は，-CS0以外の指定は，ワーニングを出力して無視します。
- ミディアム・モデルの場合は，-CSA以外の指定は，ワーニングを出力して無視します。

表5－5　ロケーション機能指定のアドレス

種別指定	説　　明
0	saddr領域を FD20-FFFFH番地に置く。
15/0FH	saddr領域を FFD20-FFFFFH番地に置く。
A	コンパイラでチェックせず，リンカでチェックする。

【使 用 例】

- saddr領域をFD20H-FFFFH番地に指定します。

```
C> cc78k4 -c4038 prime.c -mm -cs0
```

- saddr領域をFFD20H-FFFFFH番地に指定します。

```
C> cc78k4 -c4038 prime.c -ml -cs15
```

（19）機能拡張指定（-Z/-NZ）

-Z/-NZ	機能拡張指定

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，アボート・エラー（A012）となります。

【用　　途】

- 78Kシリーズ拡張機能に対し，表5－6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5－6　-Zオプションの種別指定（1/2）

種別指定	説　　明
省略時	-NZが指定されたものとみなします。
O	旧仕様（CC78K4 V1.00以前）の関数間インタフェースのコードを出力します。
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。

注　S, E, Nは同時に指定できません。

-Z/-NZ	機能拡張指定

表5ー6　-Zオプションの種別指定（2/2）

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。 （W787 Bit field type is char） ・-QC, -ZP, -ZCオプションに対し，-W2指定時，次のワーニングを出力します。 （W029 ‘ -QC ’ option is not portable） （W031 ‘ -ZP ’ option is not portable） （W032 ‘ -ZC ’ option is not portable） ・各種#pragma文に対し，-W2指定時，次のワーニングを出力します。 （W849 #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。 （W850 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。 （F801 Undefined controlなど）
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。
G	callf関数用の分岐テーブルを作成します。
H	自動的にcallf関数修飾子を付与します。

【使 用 例】

・-ZC, -ZPオプションを指定します。

```
C> cc78k4  -c4038  prime.c  -zpc
```

(20) デバイス・ファイルのサーチ・パス (-Y)

-Y	デバイス・ファイルのサーチ・パス

記述形式	-Y
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。

① ＜..¥dev＞（cc78k4.exeの起動されたパスに対して）
② CC78K4の起動されたパス
③ カレント・ディレクトリ
④ 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。したがって本コンパイラのオプション設定では本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> cc78k4  -c4038  -ya:¥tmp¥dev
```

第6章　Cコンパイラの出力ファイル

CC78K4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
- 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
- エラー・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/IV Series C Compiler V(1)x.xx Assembler Source
   ;                                        Date:(2)xxxxx Time:(3)xxxxx

   ; Command   :(4)-c4038 prime.c –sa
   ; In-file   :(5)prime.c
   ; Asm-file  :(6)prime.asm
   ; Para-file :(7)

(8) $CHGSFR(15)
   $PROCESSOR((9)4038)
(10)$DEBUG
(11)$NODEBUGA
(12)$KANJICODE SJIS
(13)$TOL_INF      03FH, 0230H, 02H, 08021H, 00H

(14)$DGS   FIL_NAM, .file,       03BH,   0FFFEH, 03FH,   067H,   01H,    00H
         :
(15)       EXTRN   @@isrem
         :
   ; line (16)1 : (17)#define  TRUE   1
   ; line (16)2 : (17)#define  FALSE  0
   ; line (16)3 : (17)#define  SIZE   200
         :
(15)_main:
(18)$DGL   1,19
(15)       push   uup
(15)       push   rp3
(15)       push   vvp
(15)       push   ax
         :
(19)??bf_main:
         :

   ; Target chip : (20)uPD784038
   ; Device file : (21)Vx.xx
```

【出力項目の説明】（1/2）

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 ‘ CC78K4 ’ 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)	SFRリロケーション情報	－	SFR リロケーション情報を出力します。 $CHGSFR（n）　n = 0, 15 0 ：sfr領域をFF00H～FFFFHへ配置 15 ：sfr領域をFFF00H～FFFFFHへ配置 $CHGSFRA A ：コンパイラでチェックせずリンカでチェックする。
(9)	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。
(10)	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。
(11)	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。 $NODEBUGAを出力します。
(12)	漢字種別情報	最大15（可変）	漢字種別を出力します。$KANJICODE SJIS, $KANJICODE EUCあるいは$KANJICODE NONE を出力します。
(13)	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（$TOL_INFで始まる情報）。
(14)	シンボル情報	－	シンボル情報を出力します（$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。
(15)	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。
(16)	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。
(17)	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。
(18)	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。

【出力項目の説明】（2/2）

項番	内　　容	桁　　数	形　　式
(19)	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。
(20)	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。
(21)	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。

6.3　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/IV Series C Compiler V①x.xx Error List       Date:②xxxxx Time:③xxxxx

Command         : ④ -c4038 prime.c –se
C-file          : ⑤prime.c
Err-file        : ⑥prime.cer
Para-file       : ⑦
*/

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

⑧char   mark[SIZE+1];

⑧main()
⑧{
⑧       int  i, prime, k, count;
⑧       cont = 0;
  *** ERROR ⑨F711 ⑩Undeclared '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);
  *** WARNING⑨W745 ⑩Expected function prototype
                 ⋮
/*
  ⑪Target chip : uPD784038
  ⑫Device file : Vx.xx
Compilation complete,    ⑬1 error(s)and    ⑭5 warning(s) found.
*/
```

【出力項目の説明】

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 ‘ CC78K4 ’ 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	4（固定）	エラー番号を ‘ #nnn ’ の形式で出力します。 #はエラーの場合はF，ワーニングの場合はWとなります。nnnはエラー番号で，10進数3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】

```
①prime.c(② 18) : ③W745 ④Expected function prototype
①prime.c(② 20) : ③W745 ④Expected function prototype
①prime.c(② 26) : ③W622 ④No return value
①prime.c(② 37) : ③W622 ④No return value
①prime.c(② 44) : ③W622 ④No return value

Target chip :⑦uPD784038
Device file :⑧Vx.xx

Compilation complete,  ⑤0 error(s)and  ⑥5 warning(s) found
```

【出力項目の説明】

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	4（固定）	エラー番号を‘ #nnn ’の形式で出力します。 #はエラーの場合はF，ワーニングの場合はWとなります。nnnはエラー番号です。
④	エラー・メッセージ	－	第9章　エラー・メッセージ参照。
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/IV Series C Compiler V①x.xx Preprocess List     Date:②xxxxx Page:③xxx

 Command       : ④ -c4038  prime.c -p -lw80
 In-file       : ⑤prime.c
 PPL-file      : ⑥prime.ppl
 Para-file     : ⑦
 */

   ⑧1 : ⑨#define TRUE   1
   ⑧2 : ⑨#define FALSE  0
   ⑧3 : ⑨#define SIZE   200
   ⑧4 : ⑨
   ⑧5 : ⑨char   mark[SIZE+1];
   ⑧6 : ⑨
/*
 ⑩Target chip : uPD784038
 ⑪Device file : Vx.xx
*/
```

【出力項目の説明】

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 ‘ CC78K4 ’ 以降を出力します。1行に入らない場合は，超過分を次行の13カラム目から出力します。1個以上の空白タブは1個の空白で置き換えます。
⑤	Cソース・モジュール・ファイル名	OSで許可している文字数	指定されたファイル名を出力します。ファイル・タイプが省略されている場合は，‘ .c ’ を付加します。1行に入らない場合は，超過分を次行の13カラム目から出力します。
⑥	プリプロセス・リスト・ファイル名	OSで許可している文字数	指定されたファイル名を出力します。ファイル・タイプが省略されている場合は，‘ .ppl ’ を付加します。1行に入らない場合は，超過分を次行の13カラム目から出力します。
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/IV Series C Compiler V①x.xx Cross reference List Date:②xxxxx Page:③xxx

 Command       : ④ -c4038  prime.c -x -lw80
 In-file       : ⑤prime.c
 Xref-file     : ⑥prime.xrf
 Para-file     : ⑦
 Inc-file      : [n] ⑧

 ATTRIB MODIFY TYPE  SYMBOL     DEFINE  REFERENCE

⑨EXTERN ⑩      ⑪array ⑫mark   ⑬5   ⑭14   ⑭16   ⑭22
⑨EXTERN ⑩      ⑪func  ⑫main   ⑬7
⑨REG1  ⑩       ⑪int   ⑫i      ⑬9   ⑭13   ⑭13   ⑭13   ⑭14
                                   ⑭15   ⑭15   ⑭15   ⑭16
                                   ⑭17   ⑭17   ⑭21
⑨REG1  ⑩       ⑪int   ⑫prime  ⑬9   ⑭17   ⑭18   ⑭21   ⑭21
⑨REG1  ⑩       ⑪int   ⑫k      ⑬9   ⑭21   ⑭21   ⑭21   ⑭22
⑨AUTO1  ⑩      ⑪int   ⑫count  ⑬9   ⑭11   ⑭19   ⑭20   ⑭25

                                  ⋮

  ⑮Target chip  : uPD784038
  ⑯ Device file : Vx.xx
```

【出力項目の説明】（1/2）

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 ‘ CC78K4 ’ 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），SREG1（sreg1･boolean1変数），OSVECT（RTOS割り込みハンドラ），TASK（RTOS用タスク関数）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。
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。

【出力項目の説明】（2/2）

項番	内　　容	桁　　数	形　　式
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

7. 1　効率良く作業する（EXITステータス機能）

CC78K4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を使用せず，コマンド行でCC78K4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】

```
cc78k4 –c4038 %1
IF ERRORLEVEL 1 GOTO ERR
cc78k4 –c4038 %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）

CC78K4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
- INC78K4　　：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）
- LIB78K4　　：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:¥cc78k4;c:¥tool
VERIFY ON
BREAK ON
SET INC78K4 = c:¥nectools32¥inc78k4
SET LIB78K4 = c:¥nectools32¥lib78k4
SET TMP = c:¥tmp
SET LANG78K = sjis
```

【説　　明】

- パス指定により，c:¥nectools32¥bin, c:¥bat, c:¥cc78k4, 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¥inc78k4から検索されます。
  Windows版は，設定しない場合，C¥NECTools32¥INC78K4（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¥lib78k4から検索されます。
  Windows版は，設定しない場合，C¥NECTools32¥LIB78K4（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使用時は，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  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

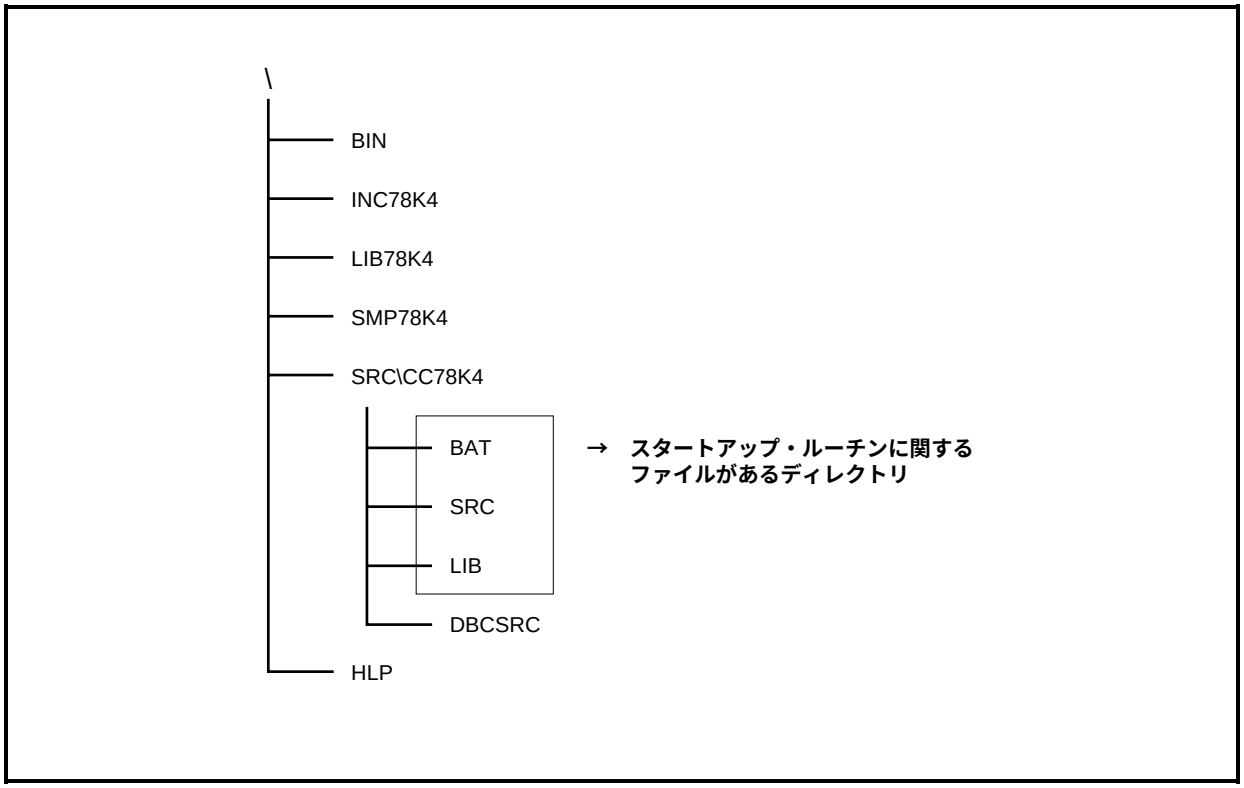
C言語によるプログラムを実行させるには，システムへ組み込むためのROM化処理，ユーザ・プログラム（main関数）の起動などを行うプログラムが必要となります。このプログラムのことをスタートアップ・ルーチンといいます。

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。CC78K4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¥CC78K4に格納されています。



次にSRC¥CC78K4以下にあるスタートアップ・ルーチンに関するファイルが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，78K4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。CC78K4が提供しているmkstup.bat（mkstup.sh）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（d4025.78k）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。

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（UNIX版は，mkstup.sh）を使用します。

また，mkstup.bat（mkstup.sh）では，RA78K4　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（mkstup.sh）のあるディレクトリSRC¥CC78K4¥BAT（src/cc78k4/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μPD784038Y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  4038Y
```

バッチ・ファイルmkstup.bat（mkstup.sh）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 ──────s4.rel
            s4b.rel
            s4c.rel
            s4cb.rel
            s4ce.rel
            s4cl.rel
            s4clb.rel
            s4cle.rel
            s4clp.rel
            s4clpb.rel
            s4clpe.rel
            s4cp.rel
            s4cpb.rel
            s4cpe.rel
            s4e.rel
            s4l.rel
            s4lb.rel
            s4le.rel
            s4lp.rel
            s4lpb.rel
            s4lpe.rel
            s4m.rel
            s4mb.rel
            s4mc.rel
            s4mcb.rel
            s4mce.rel
            s4mcl.rel
            s4mclb.rel
            s4mcle.rel
            s4me.rel
            s4ml.rel
            s4mlb.rel
            s4mle.rel
            s4p.rel
            s4pb.rel
            s4pe.rel
            s4s.rel
            s4sb.rel
            s4se.rel
            s4sl.rel
            s4slb.rel
            s4sle.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表8－4に示します。

表8－4　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。

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8－5に示します。

表8－5　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

CC78K4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8－6に示します。

表8－6　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]	s4＊.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。

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。

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インクルード・ファイルの取り込み
                                    ②ライブラリ・スイッチ
BRKSW     EQU   1   ;brk, sbrk, calloc, free, malloc, realloc function use
EXITSW    EQU   1   ;exit, atexit function use
RANDSW    EQU   1   ;rand, srand  function use
DIVSW     EQU   1   ;div          function use
LDIVSW    EQU   1   ;ldiv         function use
FLOATSW   EQU   1   ;floating point variables use

PUBLIC  _@cstart,_@cend             ③シンボル定義
$_IF(BRKSW)
   PUBLIC  _@BRKADR,_@MEMTOP,_@MEMBTM
$ENDIF
$_IF(EXITSW)
   PUBLIC  _@FNCTBL,_@FNCENT
$ENDIF
              :
                                    ④スタック解決用のシンボルの外部参照宣言
   EXTRN   _main,_@STBEG,_hdwinit
$_IF(EXITSW)
   EXTRN   _exit
$ENDIF
                                    ⑤ROM化処理用レーベルの外部参照宣言
   EXTRN   _?R_INIT,_?R_INIS,_?R_INS1,_?DATS,_?DATS1
$_IF(MEDIUM AND (LOC_0=0))
   EXTRN   _?DATA_F
$ELSE
   EXTRN   _?DATA
$ENDIF
              :
                                    ⑥標準ライブラリ用領域確保
$_IF(MEDIUM AND (LOC_0=0))
@@ DATAM DSEG   PAGE64K
$_ELSEIF(LARGE AND TWO_ALN)
@@ DATA  DSEG   UNITP
$ELSE
```

```
@@ DATA  DSEG
$ENDIF

$_IF(EXITSW)
$_IF(SMALL)
_@FNCTBL:   DS  2*32
$ELSE
_@FNCTBL:   DS  3*32
$ENDIF
_@FNCENT:   DS  2
               :
$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**

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，**RA78K4 **アセンブラ・パッケージ ユーザーズ・マニュアル 操作編（**U16708J**）を参照してください）。**

【リンク時に-sSTACK**を指定した場合の例】**

lk78k4.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，SMP78K4¥CC78K4**ディレクトリの**lk78k4.dr**を参考にしてください。**

```
                先頭アドレス    サイズ
                      ↓      ↓
  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lk78k4.drを指定します（例　-dlk78k4.dr）

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⑦～⑬について説明します。

【cstart.asmの初期設定】

```
@@VECT00    CSEG    AT  0H                    ⑦リセット・ベクタの設定
          DW  _@cstart

@@LBASE CSEG    BASE
_@cstart:
$_IF(LOC_0)
    LOCATION    0H                            ⑧ロケーション設定
$ELSE
    LOCATION    0FH
$ENDIF

$_IF(LARGE)
    SEL RB0
$ELSE
    SEL RB7
$_IF(LOC_0)
    MOV A,#00H                                ⑨汎用レジスタの初期化
$ELSE
    MOV A,#0FH
$ENDIF
    MOV V,A
    MOV U,A
    MOV T,A
$_IF(SMALL)
    MOV W,A
$ENDIF
    SEL RB6
$_IF(LOC_0)
    MOV A,#00H
$ELSE
    MOV A,#0FH
```

```
$ENDIF
   MOV V,A
   MOV U,A
   MOV T,A
$_IF(SMALL)
   MOV W,A
$ENDIF
   SEL RB0                                    ⑩レジスタ・バンクの設定
            :
$_IF(LOC_0)
   MOV A,#00H
$ELSE
   MOV A,#0FH
$ENDIF
   MOV V,A
   MOV U,A
   MOV T,A
$_IF(SMALL)
   MOV W,A
$ENDIF
$ENDIF

   MOVG   SP,#_@STBEG                         ⑪SP(スタック・ポインタ)の設定
$_IF(SMALL)
   CALL   !_hdwinit                           ⑫ハードウェア・イニシャライズ関数呼び出し
$ELSE
   CALL   !!_hdwinit                          ⑫ハードウェア・イニシャライズ関数呼び出し
$ENDIF

$_IF(BRKSW OR EXITSW OR RANDSW OR FLOATSW)    ⑬標準ライブラリ用初期値設定
   SUBW   AX,AX
$ENDIF
$_IF(BRKSW OR FLOATSW)
$_IF(SMALL)
   MOVW   !_errno,AX     ;errno <- 0
$ELSE
   MOVW   !!_errno,AX    ;errno <- 0
$ENDIF
$ENDIF
$_IF(EXITSW)
$_IF(SMALL)
   MOVW   !_@FNCENT,AX       ;FNCENT <- 0
$ELSE
   MOVW   !!_@FNCENT,AX      ;FNCENT <- 0
$ENDIF
$ENDIF

$_IF(RANDSW)
```

```
$_IF(SMALL)
    MOVW    !_@SEED+2,AX
    MOVW    !_@SEED,#1      ;SEED <- 1
$ELSE
    MOVW    !!_@SEED+2,AX
    MOVW    !!_@SEED,#1     ;SEED <- 1
$ENDIF
$ENDIF
                :
```

⑦　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  0H
                DW      _@cstart
```

⑧　ロケーション設定

コンパイル・オプション-CS**の設定に合わせて，**LOCATION 0H/0FH**を設定します。**

⑨　汎用レジスタの初期化

レジスタ・バンク0**～**7**の汎用レジスタを，次のように初期化します。**

スモール・モデル指定時は，V, U, T, W**レジスタ，ミディアム・モデルは，**V, U, T**レジスタをリセット後**1**回だけ初期化する必要があります。それらのレジスタは，その初期値をプログラム実行中保持します（ユーザが途中でそれらのレジスタを書き換えた場合，動作の保証はありません)。**

ミディアム・モデル指定時のW**レジスタ，ラージ・モデル指定時の上記すべてのレジスタについては，プログラム実行中常に変更されるため，初期化の必要はありません。**

スモール・モデル（-MS**）の場合**　：V, U, T, W**に**0**を設定**
ミディアム・モデル（-MM**）ロケーション**0H（-CS0）**の場合**　：V, U, T**に**0**を設定**
ミディアム・モデル（-MM**）ロケーション**0FH（-CS14）**の場合**　：V, U, T**に**0FH**を設定**

⑩　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。**

⑪　**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。**

⑫　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。**

⑬ **標準ライブラリ用初期値設定**

標準ライブラリ使用時に必要な初期化を行います。

（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  ⑭ROM化処理
$_IF(SMALL)
       MOVW    DE,#_@INIT
       MOVW    HL,#_@R_INIT
$ENDIF
$_IF(MEDIUM)
       MOVW    DE,#_@INIT
       MOVG    WHL,#_@R_INIT
$ENDIF
$_IF(LARGE)
       MOVG    TDE,#_@INIT
       MOVG    WHL,#_@R_INIT
$ENDIF
LINIT1:
$_IF(SMALL)
       CMPW    HL,#_?R_INIT
$ELSE
       SUBG    WHL,#_?R_INIT
$ENDIF
       BE      $LINIT2
$_IF(MEDIUM OR LARGE)
       ADDG    WHL,#_?R_INIT
$ENDIF
       MOV     A,[HL+]
       MOV     [DE+],A
       BR      $LINIT1
LINIT2:
             :
             :
```

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**）～（**f**）の**6**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
_ _sreg1 int    si1 = 0;        （e）初期値ありsreg1変数
_ _sreg1 char   sc1;            （f）初期値なしsreg1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8－1**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**），（**e**）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**），（**f**）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**8－8　**初期値の**RAM**領域（コピー先）参照）。なお，（**a**）～（**f**）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**8－7**と表**8－8**に示します。**

図8－1　ROM**化処理**

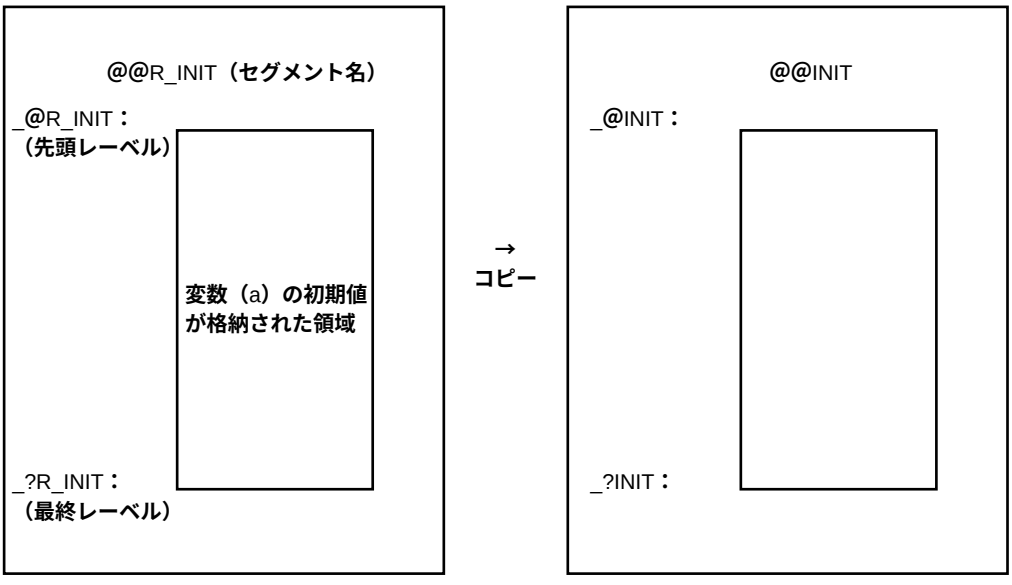


表8－7　初期値のROM領域

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
初期値ありsreg1変数（e）	@@R_INS1	_@R_INS1	_?R_INS1

表8－8　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
初期値ありsreg1変数（e）	@@INIS1	_@INIS1	_?INIS1
初期値なしsreg1変数（f）	@@DATS1	_@DATS1	_?DATS1

（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】

```
$_IF(SMALL)
       CALL    !_main          ;main();      ⑮main関数の起動
$ELSE
       CALL    !!_main         ;main();      ⑮main関数の起動
$ENDIF

$_IF(EXITSW)
       SUBW    AX,AX
$_IF(SMALL)
       CALL    !_exit          ;exit(0);     ⑯exit関数の起動
$ELSE
       CALL    !!_exit         ;exit(0);     ⑯exit関数の起動
$ENDIF
$ENDIF
       BR      $$
;
_@cend:
                                             ⑰ROM化処理で利用するセグメント，レーベルの定義
$_IF(LARGE AND TWO_ALN)
@@R_INIT       CSEG    UNITP
$_ELSEIF(SMALL)
@@RSINIT       CSEG    BASE
$ELSE
@@R_INIT       CSEG
$ENDIF
_@R_INIT:
```

```
$_IF(SMALL)
@@RSINIS       CSEG    BASE
_@R_INIS:
@@RSINS1       CSEG    BASE
_@R_INS1:
$ELSE
@@R_INIS       CSEG
_@R_INIS:
       :
$_IF(SMALL)
@@CODES CSEG    BASE
@@CALFS CSEG    FIXEDA
$ELSE
@@CODE  CSEG
@@CALF  CSEG    FIXED
$ENDIF
$_IF(MEDIUM AND (LOC_0=0))
@@CNSTM CSEG    PAGE64K
$_ELSEIF(LARGE AND TWO_ALN)
@@CNST  CSEG    UNITP
$_ELSEIF(LARGE)
@@CNST  CSEG
$ELSE
@@CNSTS CSEG    BASE
$ENDIF
@@CALT  CSEG    CALLT0
@@BITS  BSEG    SADDR2
@@BITS1 BSEG    SADDR
;
       END
```

⑮ main**関数の起動**

main**関数を呼び出します。**

⑯ 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。**

⑰ 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）～（f）**の変数**（8. 3. 2**（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。**

```
$INCLUDE (def.inc)
;
            :
       NAME   @rom

       PUBLIC  _?R_INIT,_?R_INIS,_?R_INS1
$_IF(SMALL OR LARGE)
       PUBLIC  _?INIT,_?DATA
$ENDIF
       PUBLIC  _?INIS,_?DATS
       PUBLIC  _?INIS1,_?DATS1

$_IF(LARGE AND TWO_ALN)
@@R_INIT       CSEG    UNITP   ;           ①ROM化処理で使用するレーベルの定義
$_ELSEIF(SMALL)
@@RSINIT       CSEG    BASE
$ELSE
@@R_INIT       CSEG
$ENDIF
_?R_INIT:
$_IF(SMALL)
@@RSINIS       CSEG    BASE
_?R_INIS:
@@RSINS1       CSEG    BASE
_?R_INS1:
$ELSE
@@R_INIS       CSEG
_?R_INIS:
@@R_INS1       CSEG
_?R_INS1:
$ENDIF
$_IF(LARGE AND TWO_ALN)
@@INIT  DSEG    UNITP
_?INIT:
@@DATA  DSEG    UNITP
_?DATA:
$_ELSEIF(SMALL OR LARGE)
@@INIT  DSEG
_?INIT:
@@DATA  DSEG
_?DATA:
$ENDIF
@@INIS  DSEG    SADDR2
_?INIS:
@@DATS  DSEG    SADDR2
_?DATS:
@@INIS1 DSEG    SADDR
```

```
_?INIS1:
@@DATS1 DSEG    SADDR
_?DATS1:
$ENDIF
;
       END
```

① ROM化処理で使用するレーベルの定義

ROM化処理で，（a）～（f）の変数（8. 3. 2（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

CC78K4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¥CC78K4¥BAT（src/cc78k4/bat）にあるバッチ・ファイルmkstup.bat（mkstup.sh）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8－9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8－9　ライブラリ関数で使われるシンボル

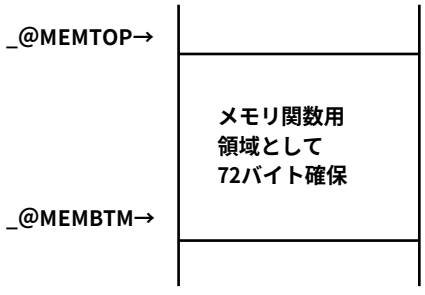
ライブラリ関数名	使われるシンボル
brk sbrk strtol strtoul malloc calloc realloc free	_errno _@MEMTOP _@MEMBTM _@BRKADR
exit	_@FNCTBL _@FNCENT
rand srand	_@SEED
div	_@DIVR
ldiv	_@LDIVR
strtok	_@TOKPTR
atof strtod **数学関数** **浮動小数点ランタイムライブラリ**	_errno

・メモリ関数で使われる領域

メモリ関数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　メモリ関数用として，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。

```
_@MEMTOP：  DS    72
_@MEMBTM：
```



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

```
_@MEMTOP：　DS　　72　→　40に変更
```

(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，RA78K4 **アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編**（U16708J）を参照してください）。

・CC78K4は，ショート・ダイレクト・アドレス領域（saddr2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，（F）FD20H-（F）FDFFHのうち任意の32バイトの領域です。

（a）norec関数の引数，自動変数（16バイト）
（b）-qr2オプションを指定した場合のregister変数（16バイト）
（c）標準ライブラリの作業用　（（b）の領域の一部）
具体的には，longjmp/setjmp使用時に_@KREG00を参照します。

・ユーザが標準ライブラリを使用しない場合，（c）の領域は使用されません。

次にリンク・ディレクティブ・ファイル（lk78k4.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。

＜lk78k4.dr＞

```
                        先頭アドレス　サイズ
memory RAM：            （0FEE00h, 20h）     → このサイズを大きくします。
memory EXTRAM：         （0FEE20h, 11E0h）     （必要に応じて，先頭アドレスも変更します。）
merge @@DATA： = EXTRAM （0F0000h, 00100h）  → 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（CC78K4 **ユーザーズ・マニュアル　言語編**（U15556J）**第11章**を参照）。

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

RX78K4およびCC78K4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，RX78K4とCC78K4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（CC78K4提供初期化処理ルーチン）に対して，startup.asm（RX78K4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，CC78K4はVer.2.40を想定しています。

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

① RX78K4で必要な次のEXTRN宣言を追加します。

【変更後】

```
        EXTRN   sys_inf,?sysrt
```

② 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
```

③　RX78K4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  0H
             DW    _@cstart
```

④　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】

```
$_IF(LARGE)
     SEL    RB0
$ELSE
     SEL    RB7
```

【変更後】

```
DI
$_IF(LARGE)
     SEL    RB0
$ELSE
     SEL    RB7
```

⑤　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】

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

⑥　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。

⑦　RX78K4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，RX78K4が制御する上で不必要となる処理も削除し，RX78K4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】

```
        $_IF(SMALL)
               CALL    !_main          ;main();
        $ELSE
               CALL    !!_main         ;main();
        $ENDIF

        $_IF(EXITSW)
               SUBW    AX,AX
        $_IF(SMALL)
               CALL    !_exit          ;exit(0);
        $ELSE
               CALL    !!_exit         ;exit(0);
        $ENDIF
        $ENDIF
               BR      $$
```

【変更後】

```
$_IF(LARGE)                    ;  ラージ・モデル
       location       0fh
       movg    tde,#sys_inf
       movw    ax,[tde]
       br      ax
$ELSE                          ;  スモール・モデル
       location       0
       sel     rb7
       mov     w,#00h
       mov     t,#00h
       mov     u,#00h
       mov     v,#00h
       sel     rb6
       mov     w,#00h
       mov     t,#00h
       mov     u,#00h
       mov     v,#00h
       sel     rb5
       mov     w,#00h
       mov     t,#00h
       mov     u,#00h
       mov     v,#00h
       sel     rb4
       mov     w,#00h
```

```
        mov     t,#00h
        mov     u,#00h
        mov     v,#00h
        sel     rb3
        mov     w,#00h
        mov     t,#00h
        mov     u,#00h
        mov     v,#00h
        sel     rb2
        mov     w,#00h
        mov     t,#00h
        mov     u,#00h
        mov     v,#00h
        sel     rb1
        mov     w,#00h
        mov     t,#00h
        mov     u,#00h
        mov     v,#00h
        sel     rb0
        mov     w,#00h
        mov     t,#00h
        mov     u,#00h
        mov     v,#00h
        movw    de,#sys_inf
        movw    ax,[de]
        br      ax
#ENDIF
```

＜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.xx   xx Xxx 20xx
;====================================
       NAME   @cstart

$INCLUDE (def.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          ;変更箇所
RANDSW   EQU   1      ;rand,srand  function use
DIVSW    EQU   1      ;div         function use
LDIVSW   EQU   1      ;ldiv        function use
STRTOKSW EQU   1      ;strtok      function use
FLOATSW  EQU   1      ;floating point variables 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,_?R_INS1,_?DATS,_?DATS1
$_IF(MEDIUM AND (LOC_0=0))
       EXTRN   _?DATA_F
$ELSE
       EXTRN   _?DATA
$ENDIF

;------------------------------------------------------------------------------
; 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
;------------------------------------------------------------------------------
$_IF(MEDIUM AND (LOC_0=0))
@@DATAM DSEG   PAGE64K
$_ELSEIF(LARGE AND TWO_ALN)
@@DATA  DSEG   UNITP
$ELSE
@@DATA  DSEG
```

```
$ENDIF

$_IF(EXITSW)
$_IF(SMALL)
_@FNCTBL:      DS     2*32
$ELSE
_@FNCTBL:      DS     3*32
$ENDIF
_@FNCENT:      DS     2
$ENDIF
$_IF(RANDSW)
_@SEED:        DS     4
$ENDIF
$_IF(DIVSW)
_@DIVR:        DS     4
$ENDIF
$_IF(LDIVSW)
_@LDIVR:       DS     8
$ENDIF
$_IF(STRTOKSW)
$_IF(SMALL OR MEDIUM)
_@TOKPTR:      DS     2
$ELSE
_@TOKPTR:      DS     3
$ENDIF
$ENDIF
$_IF(BRKSW OR FLOATSW)
_errno:        DS     2
$ENDIF
$_IF(BRKSW)
$_IF(SMALL OR MEDIUM)
_@BRKADR:      DS     2
_@MEMTOP:      DS     32
$ELSE
_@BRKADR:      DS     3
_@MEMTOP:      DS     48
$ENDIF
_@MEMBTM:
$ENDIF

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

@@LBASE CSEG    BASE
_@cstart:
$_IF(LOC_0)
       LOCATION       0H
$ELSE
```

```
       LOCATION      0FH
$ENDIF

;-------------------------------------------------------------------------------
; setting the register bank RB0 as work register set
;-------------------------------------------------------------------------------
       DI                                                    ;追加箇所
$_IF(LARGE)
       SEL    RB0
$ELSE
       SEL    RB7
$_IF(LOC_0)
       MOV    A,#00H
$ELSE
       MOV    A,#0FH
$ENDIF
       MOV    V,A
       MOV    U,A
       MOV    T,A
$_IF(SMALL)
       MOV    W,A
$ENDIF
       SEL    RB6
$_IF(LOC_0)
       MOV    A,#00H
$ELSE
       MOV    A,#0FH
$ENDIF
       MOV    V,A
       MOV    U,A
       MOV    T,A
$_IF(SMALL)
       MOV    W,A
$ENDIF
       SEL    RB5
$_IF(LOC_0)
       MOV    A,#00H
$ELSE
       MOV    A,#0FH
$ENDIF
       MOV    V,A
       MOV    U,A
       MOV    T,A
$_IF(SMALL)
       MOV    W,A
$ENDIF
       SEL    RB4
$_IF(LOC_0)
```

```
       MOV    A,#00H
$ELSE
       MOV    A,#0FH
$ENDIF
       MOV    V,A
       MOV    U,A
       MOV    T,A
$_IF(SMALL)
       MOV    W,A
$ENDIF
       SEL    RB3
$_IF(LOC_0)
       MOV    A,#00H
$ELSE
       MOV    A,#0FH
$ENDIF
       MOV    V,A
       MOV    U,A
       MOV    T,A
$_IF(SMALL)
       MOV    W,A
$ENDIF
       SEL    RB2
$_IF(LOC_0)
       MOV    A,#00H
$ELSE
       MOV    A,#0FH
$ENDIF
       MOV    V,A
       MOV    U,A
       MOV    T,A
$_IF(SMALL)
       MOV    W,A
$ENDIF
       SEL    RB1
$_IF(LOC_0)
       MOV    A,#00H
$ELSE
       MOV    A,#0FH
$ENDIF
       MOV    V,A
       MOV    U,A
       MOV    T,A
$_IF(SMALL)
       MOV    W,A
$ENDIF
       SEL    RB0
$_IF(LOC_0)
```

```
       MOV    A,#00H
$ELSE
       MOV    A,#0FH
$ENDIF
       MOV    V,A
       MOV    U,A
       MOV    T,A
$_IF(SMALL)
       MOV    W,A
$ENDIF
$ENDIF
;--------------------------------------------------------------------------
; setting the stack pointer
;
; _@STBEG is created by linker with -S option.
;--------------------------------------------------------------------------
       MOVG   SP,#_@STBEG ;SP <- stack begin address    ;必要あれば変更
$_IF(SMALL)
       CALL   !_hdwinit
$ELSE
       CALL   !!_hdwinit
$ENDIF
;--------------------------------------------------------------------------
; 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)
       SUBW   AX,AX
$ENDIF
$_IF(BRKSW OR FLOATSW)
$_IF(SMALL)
       MOVW   !_errno,AX             ;errno <- 0
$ELSE
       MOVW   !!_errno,AX            ;errno <- 0
$ENDIF
$ENDIF
$_IF(EXITSW)
$_IF(SMALL)
       MOVW   !_@FNCENT,AX           ;FNCENT <- 0
$ELSE
       MOVW   !!_@FNCENT,AX          ;FNCENT <- 0
$ENDIF
$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)
$_IF(SMALL)
       MOVW    !_@SEED+2,AX
       MOVW    !_@SEED,#1              ;SEED <- 1
$ELSE
       MOVW    !!_@SEED+2,AX
       MOVW    !!_@SEED,#1             ;SEED <- 1
$ENDIF
$ENDIF
;------------------------------------------------------------------------------
; setting _@MEMTOP address to _@BRKADR
;------------------------------------------------------------------------------
$_IF(BRKSW)
$_IF(SMALL)
       MOVW    !_@BRKADR,#_@MEMTOP     ;BRKADR <- #MEMTOP
$_ELSEIF(MEDIUM)
       MOVW    !!_@BRKADR,#_@MEMTOP    ;BRKADR <- #MEMTOP
$ELSE
       MOVG    WHL,#_@MEMTOP
       MOVG    !!_@BRKADR,WHL          ;BRKADR <- #MEMTOP
$ENDIF
$ENDIF

;------------------------------------------------------------------------------
; ROM data copy
;------------------------------------------------------------------------------
; copy external variables having initial value
$_IF(SMALL)
       MOVW    DE,#_@INIT
       MOVW    HL,#_@R_INIT
$ENDIF
$_IF(MEDIUM)
       MOVW    DE,#_@INIT
       MOVG    WHL,#_@R_INIT
$ENDIF
$_IF(LARGE)
       MOVG    TDE,#_@INIT
       MOVG    WHL,#_@R_INIT
$ENDIF
LINIT1:
$_IF(SMALL)
       CMPW    HL,#_?R_INIT
$ELSE
       SUBG    WHL,#_?R_INIT
$ENDIF
```

```
       BE     $LINIT2
$_IF(MEDIUM OR LARGE)
       ADDG   WHL,#_?R_INIT
$ENDIF
       MOV    A,[HL+]
       MOV    [DE+],A
       BR     $LINIT1
LINIT2:
; copy external variables which doesn't have initial value
$_IF(SMALL)
       MOVW   DE,#_@DATA
       MOVW   HL,#_?DATA
$ENDIF
$_IF(MEDIUM)
       MOVW   DE,#_@DATA
$_IF(LOC_0)
       MOVW   HL,#_?DATA
$ELSE
       MOVW   HL,#_?DATA_F
$ENDIF
$ENDIF
$_IF(LARGE)
       MOVG   TDE,#_@DATA
       MOVG   WHL,#_?DATA
$ENDIF
       MOV    A,#0
LDATA1:
$_IF(SMALL OR MEDIUM)
       CMPW   HL,DE
$ELSE
       SUBG   WHL,TDE
$ENDIF
       BE     $LDATA2
$_IF(LARGE)
       ADDG   WHL,TDE
$ENDIF
       MOV    [DE+],A
       BR     $LDATA1
LDATA2:
; copy sreg variables having initial value
$_IF(SMALL)
       MOVW   DE,#_@INIS
       MOVW   HL,#_@R_INIS
$ENDIF
$_IF(MEDIUM)
       MOVW   DE,#_@INIS
       MOVG   WHL,#_@R_INIS
$ENDIF
```

```
$_IF(LARGE)
       MOVG    TDE,#_@INIS
       MOVG    WHL,#_@R_INIS
$ENDIF
LINIS1:
$_IF(SMALL)
       CMPW    HL,#_?R_INIS
$ELSE
       SUBG    WHL,#_?R_INIS
$ENDIF
       BE      $LINIS2
$_IF(MEDIUM OR LARGE)
       ADDG    WHL,#_?R_INIS
$ENDIF
       MOV     A,[HL+]
       MOV     [DE+],A
       BR      $LINIS1
LINIS2:
; copy sreg variables which doesn't have initial value
$_IF(SMALL OR MEDIUM)
       MOVW    DE,#_@DATS
       MOVW    HL,#_?DATS
$ELSE
       MOVG    TDE,#_@DATS
       MOVG    WHL,#_?DATS
$ENDIF
       MOV     A,#0
LDATS1:
$_IF(SMALL OR MEDIUM)
       CMPW    HL,DE
$ELSE
       SUBG    WHL,TDE
$ENDIF
       BE      $LDATS2
$_IF(LARGE)
       ADDG    WHL,TDE
$ENDIF
       MOV     [DE+],A
       BR      $LDATS1
LDATS2:
; copy sreg1 variables having initial value
$_IF(SMALL)
       MOVW    DE,#_@INIS1
       MOVW    HL,#_@R_INS1
$ENDIF
$_IF(MEDIUM)
       MOVW    DE,#_@INIS1
       MOVG    WHL,#_@R_INS1
```

```
$ENDIF
$_IF(LARGE)
       MOVG    TDE,#_@INIS1
       MOVG    WHL,#_@R_INS1
$ENDIF
LINIS11:
$_IF(SMALL)
       CMPW    HL,#_?R_INS1
$ELSE
       SUBG    WHL,#_?R_INS1
$ENDIF
       BE      $LINIS12
$_IF(MEDIUM OR LARGE)
       ADDG    WHL,#_?R_INS1
$ENDIF
       MOV     A,[HL+]
       MOV     [DE+],A
       BR      $LINIS11
LINIS12:
; copy sreg1 variables which doesn't have initial value
$_IF(SMALL OR MEDIUM)
       MOVW    DE,#_@DATS1
       MOVW    HL,#_?DATS1
$ELSE
       MOVG    TDE,#_@DATS1
       MOVG    WHL,#_?DATS1
$ENDIF
       MOV     A,#0
LDATS11:
$_IF(SMALL OR MEDIUM)
       CMPW    HL,DE
$ELSE
       SUBG    WHL,TDE
$ENDIF
       BE      $LDATS12
$_IF(LARGE)
       ADDG    WHL,TDE
$ENDIF
       MOV     [DE+],A
       BR      $LDATS11
LDATS12:

;-------------------------------------------------------------------------
; branches to the reset routine for system initialization of RX78K/IV
;-------------------------------------------------------------------------
$_IF(LARGE)                    ;ラージ・モデル              ;
       movg    tde,#sys_inf                              ;
       movw    ax,[tde]                                  ;
```

```
        br      ax                                      ;
$ELSE                           ;スモール・モデル       ;
        sel     rb7                                     ;
        mov     w,#00h                                  ;
        mov     t,#00h                                  ;
        mov     u,#00h                                  ;
        mov     v,#00h                                  ;
        sel     rb6                                     ;
        mov     w,#00h                                  ;
        mov     t,#00h                                  ;
        mov     u,#00h                                  ;
        mov     v,#00h                                  ;
        sel     rb5                                     ;
        mov     w,#00h                                  ;
        mov     t,#00h                                  ;
        mov     u,#00h                                  ;
        mov     v,#00h                                  ;
        sel     rb4                                     ;
        mov     w,#00h                                  ;変更箇所
        mov     t,#00h                                  ;
        mov     u,#00h                                  ;
        mov     v,#00h                                  ;
        sel     rb3                                     ;
        mov     w,#00h                                  ;
        mov     t,#00h                                  ;
        mov     u,#00h                                  ;
        mov     v,#00h                                  ;
        sel     rb2                                     ;
        mov     w,#00h                                  ;
        mov     t,#00h                                  ;
        mov     u,#00h                                  ;
        mov     v,#00h                                  ;
        sel     rb1                                     ;
        mov     w,#00h                                  ;
        mov     t,#00h                                  ;
        mov     u,#00h                                  ;
        mov     v,#00h                                  ;
        sel     rb0                                     ;
        mov     w,#00h                                  ;
        mov     t,#00h                                  ;
        mov     u,#00h                                  ;
        mov     v,#00h                                  ;
        movw    de,#sys_inf                             ;
        movw    ax,[de]                                 ;
        br      ax                                      ;
$ENDIF                                                  ;
;
_@cend:
```

```
;------------------------------------------------------------------------------
; define segment and label used by ROMable processing
;------------------------------------------------------------------------------
$_IF(LARGE AND TWO_ALN)
@@R_INIT      CSEG   UNITP
$_ELSEIF(SMALL)
@@RSINIT      CSEG   BASE
$ELSE
@@R_INIT      CSEG
$ENDIF
_@R_INIT:
$_IF(SMALL)
@@RSINIS      CSEG   BASE
_@R_INIS:
@@RSINS1      CSEG   BASE
_@R_INS1:
$ELSE
@@R_INIS      CSEG
_@R_INIS:
@@R_INS1      CSEG
_@R_INS1:
$ENDIF
$_IF(MEDIUM AND (LOC_0=0))
@@INITM DSEG   PAGE64K
_@INIT:
@@DATAM DSEG   PAGE64K
_@DATA:
$_ELSEIF(LARGE AND TWO_ALN)
@@INIT  DSEG   UNITP
_@INIT:
@@DATA  DSEG   UNITP
_@DATA:
$ELSE
@@INIT  DSEG
_@INIT:
@@DATA  DSEG
_@DATA:
$ENDIF
@@INIS  DSEG   SADDR2
_@INIS:
@@DATS  DSEG   SADDR2
_@DATS:
@@INIS1 DSEG   SADDR
_@INIS1:
@@DATS1 DSEG   SADDR
_@DATS1:
$_IF(SMALL)
@@CODES CSEG   BASE
```

```
@@CALFS CSEG   FIXEDA
$ELSE
@@CODE  CSEG
@@CALF  CSEG   FIXED
$ENDIF
$_IF(MEDIUM AND (LOC_0=0))
@@CNSTM CSEG   PAGE64K
$_ELSEIF(LARGE AND TWO_ALN)
@@CNST  CSEG   UNITP
$_ELSEIF(LARGE)
@@CNST  CSEG
$ELSE
@@CNSTS CSEG   BASE
$ENDIF
@@CALT  CSEG   CALLT0
@@BITS  BSEG   SADDR2
@@BITS1 BSEG   SADDR
;
      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8－10　初期化データのROM領域のセクション

メモリ・モデル	変数の種類	セグメント	先頭レーベル	最終レーベル
ラージ・モデル（2バイト・アラインメントなし），ミディアム・モデル	初期値あり外部変数（a）	@ER_INIT CSEG	E@R_INIT	E?R_INIT
	初期値ありsreg変数（c）	@ER_INIS CSEG	E@R_INIS	E?R_INIS
	初期値ありsreg1変数（e）	@ER_INS1 CSEG	E@R_INS1	E?R_INS1
ラージ・モデル（2バイト・アラインメントあり）	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	E@R_INIS	E?R_INIS
	初期値ありsreg1変数（e）	@ER_INS1 CSEG	E@R_INS1	E?R_INS1
スモール・モデル	初期値あり外部変数（a）	@ERSINIT CSEG BASE	E@R_INIT	E?R_INIT
	初期値ありsreg変数（c）	@ERSINIS CSEG BASE	E@R_INIS	E?R_INIS
	初期値ありsreg1変数（e）	@ERSINS1 CSEG BASE	E@R_INS1	E?R_INS1

表8－11　コピー先のRAM領域のセクション

メモリ・モデル	変数の種類	セグメント	先頭レーベル	最終レーベル
ラージ・モデル（2バイト・アラインメントなし），ミディアム・モデル（ロケーション0），スモール・モデル	初期値あり外部変数（a）	@EINIT DSEG	E@INIT	E?INIT
	初期値なし外部変数（b）	@EDATA DSEG	E@DATA	E?DATA
	初期値ありsreg変数（c）	@EINIS DSEG SADDR2	E@INIS	E?INIS
	初期値なしsreg変数（d）	@EDATS DSEG SADDR2	E@DATS	E?DATS
	初期値ありsreg1変数（e）	@EINIS1 DSEG SADDR	E@INIS1	E?INIS1
	初期値なしsreg1変数（f）	@EDATS1 DSEG SADDR	E@DATS1	E?DATS1
ラージ・モデル（2バイト・アラインメントあり）	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 SADDR2	E@INIS	E?INIS
	初期値なしsreg変数（d）	@EDATS DSEG SADDR2	E@DATS	E?DATS
	初期値ありsreg1変数（e）	@EINIS1 DSEG SADDR	E@INIS1	E?INIS1
	初期値なしsreg1変数（f）	@EDATS1 DSEG SADDR	E@DATS1	E?DATS1
ミディアム・モデル（ロケーション15）	初期値あり外部変数（a）	@EINITM DSEG PAGE64K	E@INIT	E?INIT_F
	初期値なし外部変数（b）	@EDATAM DSEG PAGE64K	E@DATA	E?DATA_F
	初期値ありsreg変数（c）	@EINIS DSEG SADDR2	E@INIS	E?INIS
	初期値なしsreg変数（d）	@EDATS DSEG SADDR2	E@DATS	E?DATS
	初期値ありsreg1変数（e）	@EINIS1 DSEG SADDR	E@INIS1	E?INIS1
	初期値なしsreg1変数（f）	@EDATS1 DSEG SADDR	E@DATS1	E?DATS1

・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@R_INS1, E@INIT, E@DATA, E@INIS, E@DATS, E@INIS, E@DATS1

・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?R_INS1, E?INIT（ミディアム・モデル・ロケーション15はE?INIT_F）, E?DATA（ミディアム・モデル・ロケーション15はE?DATA_F）, E?INIS, E?DATS, E?INIS1, E?DATS1

・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（ミディアム・モデル・ロケーション15はE?DATA_F）まで，E@DATSからE?DATSまで，およびE@DATS1からE?DATS1まで0を埋め込みます。

第9章　エラー・メッセージ

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）　：F712 Declaration syntax
prime.c（8）　：F301 Syntax error
prime.c（8）　：F701 External definition syntax
prime.c（19）：W745 Expected function prototype

ただし，F101, F103, F104の内部エラーのみ次の出力形式となります。

```
[xxx.c <yyy> zzz] F101 Internal error
[xxx.c <yyy> zzz] F103 Intermediate file error
[xxx.c <yyy> zzz] F104 Illegal use of register
```

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。

```
A/F/Wnnn
```

A：ABORT

エラー・メッセージ出力後，ただちにコンパイル処理を終了します。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・ファイルは，出力されません。

F：FATAL

エラー・メッセージ出力後，エラー部分を無視し処理を続行します。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・ファイルは出力されません。

W：WARNING

ワーニング・メッセージを出力後，処理を続行します。オプションで指定したファイルが出力されます。

nnn（3桁の数字）

001～	コマンド行に対するエラー・メッセージ
101～	内部エラー，メモリに対するエラー・メッセージ
201～	文字に対するエラー・メッセージ
301～	構成要素に対するエラー・メッセージ
401～	変換に対するエラー・メッセージ
501～	式に対するエラー・メッセージ
601～	文に対するエラー・メッセージ
701～	宣言，関数定義に対するエラー・メッセージ
801～	前処理指令に対するエラー・メッセージ
901～	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。

(1) コマンド行に対するエラー・メッセージ〈001～〉 (1/3)

A001	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 'cc78k4 –' 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。
A002	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 'cc78k4 –' 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。
A003	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。
A004	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。
A005	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。
A006	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。
A007	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。
A008	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。
A009	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。
A010	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。
A011	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。
A012	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 'cc78k4 –' 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。
A013	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 'cc78k4 --' 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。
A014	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 'cc78k4 --' 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。
A015	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。

（1）コマンド行に対するエラー・メッセージ《001～》（2/3）

A016	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 ‘cc78k4 –’ 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。
A017	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。
A018	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 ‘cc78k4 --’ 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。
A019	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。
A020	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。
A021	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。
W022	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。
W023	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。
A024	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。
W027	メッセージ	'-MS' or '-MM' option specified - ignored '-CSA'
	原因	メモリ・モデル指定オプションでスモール・モデル（-MS）またはミディアム・モデル（-MM）が指定されたため，ロケーション機能指定オプション-CSAは無視されます。
W028	メッセージ	'-MS' option specified - ignored '-CS15'
	原因	メモリ・モデル指定オプションでスモール・モデル（-MS）が指定されたため，ロケーション機能指定オプション-CS15は無視されます。
W029	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）。
W031	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）。
W032	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）。

（1）コマンド行に対するエラー・メッセージ《001～》（3/3）

A033	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 ‘cc78k4 –’ 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。
W042	メッセージ	'-QH' option is not portable
	原因	-QHオプションはANSI準拠ではありません（-QH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）。
W043	メッセージ	‘-ZO’ option specified - ignored ‘-ZR’
	原因	旧仕様関数インタフェース指定オプション（-ZO）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。
W046	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。

(2) 内部エラー，メモリに対するエラー・メッセージ〈101～〉

F101	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。
F102	メッセージ	Too many errors
	原因	FATAL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。
F103	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。
F104	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。
F105	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。
A106	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。
F108	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。
F109	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。
F110	メッセージ	Compiler limit：too much code defined in file
	原因	●スモール・モデルの場合 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。
		●ミディアム／ラージ・モデルの場合 ファイル内のコードに割り当てられた領域が，1024 Kバイトの制限を越えました。
F111	メッセージ	Compiler limit：too much global data defined in file
	原因	●スモール／ミディアム・モデルの場合 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。
		●ラージ・モデルの場合 ファイル内のグローバル変数に割り当てられた領域が16 Mバイトの制限を越えました。
F113	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。

（3）文字に対するエラー・メッセージ《201～》

F201	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。
F202	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。
W203	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。

（4）構成要素に対するエラー・メッセージ《301～》（1/2）

F301	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。
F303	メッセージ	Expected identifier
	原因	goro文の識別子が必要です。
	処理	goro文に指定する識別子を正しく記述してください。
W304	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。
F305	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。
F306	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。
F307	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。
F308	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。
W309	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。
W310	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。
F311	メッセージ	Number syntax
	原因	定数の表現が誤っています。
F312	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。
F313	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。
F314	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。
F315	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。

（4）構成要素に対するエラー・メッセージ《301～》（2/2）

F316	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。
F317	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。
F318	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。
F319	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。
W320	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。
F321	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。
F322	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。
F323	メッセージ	Missing‘区切り子’
	原因	区切り子に誤りがあります。
F324	メッセージ	Too many }'s
	原因	‘｛’と‘｝’が正しく対応していません。
F325	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。
F326	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。
F327	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。
W328	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。
F329	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。
F336	メッセージ	'-ZO' option specified - _ _flash keyword is not allowed
	原因	旧仕様関数インタフェース指定オプション（-ZO）が指定されたため，_ _flashキーワードは使用できません。
W337	メッセージ	‘-ZO’ option specified –ignored ‘_ _pascal’ in this file
	原因	旧仕様関数インタフェース指定オプション（-ZO）が指定されたため，このファイルでは_ _pascalキーワードは無視されます。
W340	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。

（5）変換に対するエラー・メッセージ〈401～〉

W401	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。
F402	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。
F403	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。
F404	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。
F405	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。
F406	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。
F407	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。
F408	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。
F409	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。
W410	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。
W411	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。
W412	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。
W413	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。
F414	メッセージ	Expected pointer
	原因	ポインタが必要です。

（6）式に対するエラー・メッセージ〈501～〉（1/4）

F501	メッセージ	Expression syntax
	原因	式の構文エラーが起こりました。
F502	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。
W503	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。
W504	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。
W505	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。

（6）式に対するエラー・メッセージ《501～》（2/4）

F507	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。
W508	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。
W509	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。
W510	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。
W511	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。
F512	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。
F513	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。
F514	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。
F515	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。
F516	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。
F517	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。
F518	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。
F519	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。
F520	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。
F521	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。
W522	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。
F523	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。

（6）式に対するエラー・メッセージ〈501～〉（3/4）

F524	メッセージ	‘ & ’ on constant
	原因	定数のアドレスは得られません。
F525	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。
F526	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。
F527	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。
W528	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。
F529	メッセージ	Sizeof returns zero
	原因	sizeof式の値が0になっています。
F530	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。
F531	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。
F532	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。
F533	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。
F534	メッセージ	Divide check
	原因	／演算，%演算の除数が0です。
F535	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。
F536	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。
F537	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。
F538	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。
F539	メッセージ	Expected constant expression
	原因	定数式が必要です。
W540	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。
F541	メッセージ	Function argument has void type
	原因	関数の引数がvoid型です。
W543	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。

（6）式に対するエラー・メッセージ《501～》（4/4）

F544	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。
F546	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。

（7）文に対するエラー・メッセージ《601～》（1/2）

F602	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。
F603	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。
F604	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。
F605	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。
F606	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。
F607	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。
F608	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。
F609	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。
F610	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。
F611	メッセージ	Inappropriate ‘ else ’
	原因	ifとelseの対応がとれていません。
W613	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。
W615	メッセージ	Statement not reached
	原因	絶対に到達しない文があります。
F617	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。
F620	メッセージ	Break/continue error
	原因	break, continue文の位置が誤っています。
F621	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。
W622	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。

（7）文に対するエラー・メッセージ〈601～〉（2/2）

F623	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。

（8）宣言，関数定義に対するエラー・メッセージ〈701～〉（1/5）

F701	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。
F702	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。
F703	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。
F704	メッセージ	Function returns illegal type
	原因	関数の返り値が不正な型です。
F705	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。
F706	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。
F707	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。
W708	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。
F710	メッセージ	Illegal storage class
	原因	関数の外部でauto, register宣言が行われているか，または，関数内でboolean変数が定義されています。
F711	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。
F712	メッセージ	Declaration syntax
	原因	宣言文が文法に合っていません。
F713	メッセージ	Redefined ‘ 変数名 ’
	原因	同じ変数が2回以上定義されています。
	処理	変数の定義は1回にしてください。
W714	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，言語編（U15556J）第11章を参照してください。
F715	メッセージ	Too many sreg variables
	原因	sreg変数の宣言が多すぎます。
F716	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。

（8）宣言，関数定義に対するエラー・メッセージ〈701～〉（2/5）

F717	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，言語編（U15556J）第11章を参照してください。
F718	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，言語編（U15556J）第11章を参照してください。
F719	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。
F720	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。
W721	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。
F723	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。
F724	メッセージ	Structure redefinition
	原因	同じ構造体が再定義されています。
W725	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。
	処理	構造体のメンバに配列を使用し，添え字を定数演算で与えている場合は，-QC2の作用によりオーバフローして領域を確保されない場合があります。その場合は，-QC1/-QCのように 指定してください。-QCはディフォールト・オプションです。
F726	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。
F727	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。
F728	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。
F729	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。
F730	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。
F731	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。
F732	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。
F733	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。
F734	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。

（8）宣言，関数定義に対するエラー・メッセージ〈701～〉（3/5）

F735	メッセージ	Illegal enumeration constant
	原因	列挙定数が不正です。
F736	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。
F737	メッセージ	Undeclared structure/union/enum tag
	原因	タグが宣言されていません。
F738	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。
F739	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。
F740	メッセージ	Negative subscript
	原因	配列の大きさの指定が負です。
F741	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。
F742	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。
F743	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。
W744	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。
W745	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。
F747	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。
W748	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。
W749	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。
F750	メッセージ	Initializer syntax
	原因	初期化が文法にあっていません。
F751	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。
W752	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。
F753	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。
F756	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。
F757	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。

（8）宣言，関数定義に対するエラー・メッセージ〈701～〉（4/5）

F758	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。
F759	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。
W760	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。
W761	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。
W762	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。
W763	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。
W764	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。
F765	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。
F766	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。
F767	メッセージ	Too many _ _sreg1 variables
	原因	_ _sreg1変数の宣言が多すぎます。
F768	メッセージ	Too many _ _boolean1 type variables
	原因	_ _boolean1型変数の宣言が多すぎます。
F770	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。
F771	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。
F772	メッセージ	Callt/callf/noauto/norec/_ _pascal are not allowed for interrupt function
	原因	interrupt関数は，callt, callf, noauto, norec, _ _pascal宣言は指定できません。
F773	メッセージ	Cannot call interrupt function
	原因	interrupt関数をコールできません。
F774	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。
F775	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。
F776	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。
F777	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。
F778	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。

（8）宣言，関数定義に対するエラー・メッセージ〈701～〉（5/5）

W779	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。
F780	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。
W787	メッセージ	Bit field type is char
	原因	ビット・フィールドの型にchar型を指定しています。
F788	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。
F789	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。
F790	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。
F791	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。
W792	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。
W793	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。
F799	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）前処理指令に対するエラー・メッセージ〈801～〉（1/5）

F801	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。
F802	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。
F803	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。
W804	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。
F805	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。
F806	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。
F807	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。

（9）前処理指令に対するエラー・メッセージ〈801～〉（2/5）

F810	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INC78K4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。
F811	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。
F812	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。
F813	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。
F814	メッセージ	Illegal macro name
	原因	マクロ名が正しくありません。
F815	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。
W816	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。
W817	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。
F818	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。
W819	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。
F821	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。
F822	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。
F823	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。
W824	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。
W825	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。
F826	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。
F827	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
F828	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。
W829	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。

（9）前処理指令に対するエラー・メッセージ《801～》（3/5）

F830	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。
F831	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。
W832	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。
F833	メッセージ	Expected #asm
	原因	#asmがありません。
F834	メッセージ	Expected #endasm
	原因	#endasmがありません。
W835	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。
W836	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。
W837	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。
F838	メッセージ	Duplicated pragma VECT or INTERRUPT or RTOS_INTERRUPT ‘ 文字列 ’
	原因	#pragma VECT ‘ 文字列 ’ または INTERRUPT ‘ 文字列 ’ またはRTOS_INTERRUPT ‘ 文字列 ’ が重複しています。
F839	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 ’ またはINTERRUPT ‘ 文字列 ’ またはRTOS_INTERRUPT ‘ 文字列 ’ があります。
W840	メッセージ	Undefined interrupt function ‘ 関数名 ’ - ignored BANK or SP_SWITCH specified
	原因 コンパイラ	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。 レジスタ・バンク指定，スタック切り替え指定を無視します。
F842	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。
F843	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。
F845	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。
W846	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。
F847	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。

（9）前処理指令に対するエラー・メッセージ《801～》（4/5）

F848	メッセージ	Cannot allocate segment to saddr area with -CSA‘ セクション名 ’
	原因	-CSA指定時はセクションをアドレス指定してsaddr領域に配置できません。
	処理	セクションをアドレス指定してsaddr領域に配置する場合は，-CS0または-CS15オプションのどちらかを指定してください。
W849	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。
W850	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。
W851	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。
W852	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。
F853	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。
F854	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。
F855	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。
W856	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。
W857	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。
F858	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。
F859	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。
F860	メッセージ	Redefined #pragma EXT_TABLE
	原因	#pragma EXT_TABLEが再定義されています。
F861	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。
F862	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。
F863	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。
F864	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。
F865	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。

（9）前処理指令に対するエラー・メッセージ〈801～〉（5/5）

F866	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。
F867	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。
F868	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。
W869	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。
W870	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。
W871	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。
F899	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。

（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〈901～〉（1/2）

A901	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。
A902	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. 3（20）デバイス・ファイルのサーチ・パスを参照してください。
A903	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。
A904	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。
A905	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。
A906	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。
A907	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。
A908	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。
A909	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。

（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〈901～〉（2/2）

A910	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。
A911	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。
A912	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。
A913	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。
A914	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。
W915	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。W837の処理を行ってください。
F922	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。
A923	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

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(s, i)
char *s;
int i;

{
        int j;
        char *ss;
```

```
        j = i;
        ss = s;
}

putchar(c)
char c;
{

        char d;
        d = c;
}
```

A. 2　実 行 例

```
C>cc78K4 –c4026 prime.c -a –p -x -e -ng
```

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

sample¥prime.c(18):W745 Expected function prototype
sample¥prime.c(20):W745 Expected function prototype
sample¥prime.c(26):W622 No return value
sample¥prime.c(37):W622 No return value
sample¥prime.c(44):W622 No return value

Target chip:uPD784026
Device file:Vx.xx

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

A. 3　出力リスト

(1) アセンブラ・ソース・モジュール・ファイル

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

; Command   : -c4026 prime.c -a -p -x -e -ng
; In-file   : prime.c
; Asm-file  : prime.asm
; Para-file :

$CHGSFR(15)
$PROCESSOR(4026)
$NODEBUG
$NODEBUGA
$KANJICODE SJIS
$TOL_INF       03FH, 0230H, 02H, 08021H, 00H

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

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

@@DATA  DSEG
_mark:  DS      (201)

; line    5
; line    8

@@CODE  CSEG
_main:
       push    uup
       push    rp3
       push    vvp
       push    ax
; line   11
       subw    ax,ax
       movw    [sp+0],ax       ; count
; line   13
       subw    rp3,rp3
L0003:
       cmpw    rp3,#0C8H       ; 200
       bgt     $L0004
; line   14
       movw    de,rp3
       mov     a,#01H  ; 1
       mov     _mark[de],a
       incw    rp3
       br      $L0003
```

```
L0004:
; line   15
       subw    rp3,rp3
L0006:
       cmpw    rp3,#0C8H       ; 200
       bgt     $L0007
; line   16
       movw    de,rp3
       mov     a,_mark[de]
       cmp     a,#00H  ; 0
       be      $L0015
; line   17
       addw    de,de
       addw    de,#03H ; 3
       movw    up,de
; line   18
       push    up
       movg    whl,#L0011
       call    $!_printf
       pop     ax
; line   19
       movw    ax,[sp+0]       ; count
       incw    ax
       movw    [sp+0],ax       ; count
; line   20
       movw    bc,ax
       movw    ax,#08H ; 8
       call    !!@@isrem
       or      a,x
       bne     $L0012
       mov     x,#0AH  ; 10
       call    $!_putchar
L0012:
; line   21
       movw    ax,rp3
       addw    ax,up
       movw    vp,ax
L0014:
       cmpw    vp,#0C8H        ; 200
       bgt     $L0015
; line   22
       movw    hl,vp
       mov     a,#00H  ; 0
       mov     _mark[hl],a
       addw    vp,up
       br      $L0014
L0015:
; line   24
       incw    rp3
       br      $L0006
L0007:
; line   25
       movw    ax,[sp+0]       ; count
       push    ax
       movg    whl,#L0017
       call    $!_printf
       pop     ax
; line   26
       pop     ax
```

```
       pop    vvp
       pop    rp3
       pop    uup
       ret
; line   31
_printf:
       push   uup
       push   rp3
       push   vvp
       movg   uup,whl
; line   35
       movw   ax,[sp+11]     ; i
       movw   rp3,ax
; line   36
       movg   vvp,uup
; line   37
       pop    vvp
       pop    rp3
       pop    uup
       ret
; line   41
_putchar:
       push   rp3
       mov    r6,x
; line   43
       mov    r7,r6
; line   44
       pop    rp3
       ret
       END


; Target chip : uPD784026
; Device file : Vx.xx
```

(2) プリプロセス・リスト・ファイル

```
/*
78K/IV Series C Compiler Vx.xx Preprocess List                Date:xx xxx xxxx Page:1

Command  : -c4026 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 : uPD784026
 Device file : Vx.xx
*/
```

(3) クロスレファレンス・リスト・ファイル

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

Command  :-c4026 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
REG1          int     prime           9      17     18     21     21
REG1          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
PARAM         int     i              30      35
REG1          int     j              32      35
REG1          pointer ss             33      36
REG1          char    c              40      43
PARAM
REG1          char    d              42      43
              #define TRUE            1      14
              #define FALSE           2      22
              #define SIZE            3       5     13     15     21


Target chip:uPD784026
Device file:VX.XX
```

(4) エラー・リスト・ファイル

```
PRIME.C(   18):W745 Expected function prototype
PRIME.C(   20):W745 Expected function prototype
PRIME.C(   26):W622 No return value
PRIME.C(   37):W622 No return value
PRIME.C(   44):W622 No return value



Target chip:uPD784026
Device file:Vx.xx
Compilation complete,    0 error(s) and    5 warning(s) found.
```

付録B　使用上の注意事項一覧

項番	注意事項
1	【オプション指定に関する注意事項】 (a) 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。 (b) -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，アボート・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。 (c) 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】 CC78K4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。 (a) 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。 (b) 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，大括弧（[ ]）を含むファイル名，パス名，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。 (c) 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。 (d) 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。 (e) ID78K4/ID78K4-NS,　SM78K4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），英大数字（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。
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。 (a) #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8文字以内のシンボルを使用しない場合アセンブル時にアボートA114が出力されます。

<table>
<tr><th>項番</th><th>注意事項</th></tr>
<tr><td>6</td><td>(b) 『通常の関数』，『callf関数』，『callt関数と割り込み関数』の定義は，この3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W717が出力されます。
(c) 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。
(d) 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。
(e) 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RA78K4を起動します。
(f) #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。アセンブル時にアボートA106が出力されます。</td></tr>
<tr><td>7</td><td>【コンパイラ・オプション-QC2指定時の注意事項】
CC78K4では，オプション-QC2指定時に，定数と文字定数の型を表現できる範囲により次のように取り扱っています。
<table><tr><td>-128〜+127</td><td>char型</td></tr><tr><td>128〜255</td><td>unsigned char型</td></tr><tr><td>0U〜255U</td><td>unsigned char型</td></tr><tr><td>256〜</td><td>int型</td></tr><tr><td>〜-129</td><td>int型</td></tr><tr><td>'¥0'〜'¥377'</td><td>char型</td></tr></table>
-QC2オプション指定時には，char型定数どうし，または，unsigned char型定数どうしの演算結果は，それぞれchar型，unsigned char型として扱われます。また，char型定数とunsigned char型定数との演算結果は，unsigned char型として扱われます。
演算結果がオーバフローする場合には，定数のどちらかを表現できる型でキャストするか，-QC1または-QC（ディフォールト）オプションを同時に指定してください。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。

例　–QC2オプション指定時
<table><tr><td>int i;</td><td></td></tr><tr><td>i = 20*20</td><td>/* 負の値 */</td></tr><tr><td>i = (int)20*20</td><td>/* 400 */</td></tr></table>
備考　ただし，-QUオプション指定時には，すべてのchar型データがunsigned char型として扱われます。‘¥200’〜‘¥377’の範囲の文字定数の場合は，unsigned char型として扱われ，＋128〜＋255の値を持ちます。</td></tr>
<tr><td>8</td><td>【利用可能なアセンブラ・パッケージ】
ロング・ファイル名に対応しているため，Ver.1.50より前のRA78K4を使用するとエラーになる場合があります。</td></tr>
<tr><td>9</td><td>【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。</td></tr>
</table>

項番	注意事項

10	**【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，RA78K4　アセンブラ・パッケージ　ユーザーズ・マニュアル　操作編（U16708J）およびコンパイラに添付されているlk78k4.dr（SMP78K4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**
	＜lk78k4.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・シンボルの自動生成オプション-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lk78k4.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）
	`> lk78k4 s4l.rel prime.rel –bcl4.lib -SSTK –Dlk78k4.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。**
	`“***ERROR F206 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。

項番	注意事項
10	【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。 1. 配置できないセグメントのサイズを調べる（.mapファイル参照）。 2. 手順1で調べたセグメントのサイズをもとに，ディレクティブ・ファイルでセグメントが配置されている領域のサイズを拡大する。 3. ディレクティブ・ファイル指定オプション（-D）を指定してリンクする。 ただし，手順1ではエラーで指摘されたセグメントの種類により，次のようにセグメントのサイズを調べる方法が異なります。 (1) 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。 (2) 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。
11	【va_startマクロ使用時の注意事項】 -ZOオプション未指定時，関数により第一引数のオフセットが異なるため，stdarg.hに定義されているva_startマクロの動作は保証されません。
12	【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。
13	【スタートアップ・ルーチン，ライブラリについて】 (a) 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（cc78k4.exeまたはcc78k4）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。 (b) 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。
14	【-ZOオプションについて】 CC78K4 Ver.1.00で開発したソースの場合，アセンブラを併用している場合には，-ZO オプションを指定しない限り，書き換えが必要になる場合があります。 ただし，-ZO オプションを指定すると，コード効率は低下し，CC78K4 Ver.2.00以上の本来の性能は出ません。
15	【ID78K4, ID78K4-NS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。コンパイル・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。 また，コンパイル・オプション-QL4を指定したオブジェクト・モジュール・ファイルを含むロード・モジュール・ファイルをディバグする際には，コンパイラが提供するランタイム・ライブラリ・ソースをカレントにおいてください。 このロード・モジュール・ファイルをディバグ中，NextやStepコマンド指定時に，ランタイム・ライブラリのウインドウが現れ，カレントPCがランタイム・ライブラリ側に移る場合があります。この場合は，Cソースの次の行に break pointを設け，Goコマンドで呼び出し関数側に戻ってください。

項番	注意事項
16	【SM78K4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。 コンパイル・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。 また，コンパイル・オプション-QL4を指定したオブジェクト・モジュール・ファイルを含むロード・モジュール・ファイルをディバグする際には，コンパイラが提供するランタイム・ライブラリ・ソースをカレントにおいてください。 このロード・モジュール・ファイルをディバグする場合，Nextコマンドを行うと暴走する場合があるので，Nextコマンドを行わないでください。 また，Stepコマンド指定時に，ランタイム・ライブラリのウインドウが現れ，カレントPCがランタイム・ライブラリ側に移る場合があります。この場合は，Cソースの次の行に break pointを設け，Goコマンドで呼び出し関数側に戻ってください。
17	【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。CC78K4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8－7を参照してください。 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　：S4.REL (2) C標準ライブラリ用の領域を使用する場合　　：S4L.REL 【使用例】 C:> LK78K4 S4.REL SAMPLE.REL -S –BCL4.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 S4.REL　　：スタートアップ・ルーチン CL4.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，CC78K4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。 ・CC78K4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（CL4*F.LIB）を使用する場合は，通常のライブラリ（CL4*.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 –BCL4F.LIB –BCL4.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 –BCL4.LIB –BCL4F.LIB

項番	注意事項
18	【スタック領域用シンボル生成（-S）について】 CC78K4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。
19	【ROMコードについて】 ROMコード発注をする場合には，-Rおよび-Uオブジェクトコンバータ・オプションを指定してください （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充填値　：指定された充填値をROMコードの空き領域に埋め込みます。
20	【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。
21	【-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。
22	【シンボル名長に関する注意事項】 ID78K4-NS V1.11，ID78K4 V1.42, SM78K4 V1.42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。

23	【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。
	・品種指定オプション(-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"`
	`*/`

<table>
<tr><th>項番</th><th>注意事項</th></tr>
<tr><td>23</td><td>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・ウィ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。
(e) 大括弧（[ ]）を含むファイル名，パス名は扱えません。</td></tr>
<tr><td>24</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（F301，F701）となります。
例
<pre>f (void) ; /* F301 : Syntax error */
              /* F701 : 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>25</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]   /* F712などのエラー */</pre>【回避策】 ありません。</td></tr>
<tr><td>26</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(F803，F814，F821など)となります。
例
<pre>/* com1 */ #pragma sfr   /* F803 */
/* com2 */ #define ONE 1   /* F803 */
#define /* com3 */ TWO 2   /* F814 */
#ifdef /* com4 */ ANSI_C   /* F814 */

/* com5 */ #endif
#define SUB(p1, /* com6 */ p2) p2 = p1   /* F821 */</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>

項番	注意事項

27	【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，(1) の条件を満たす場合はワーニング，(2) の条件を満たす場合はエラーとなります。
	(1) 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W510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 );  /* W510 Pointer mismatch */`
	`}`
	(2) 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F737となります。
	例
	`void func1( int , struct st ) ;  /* F737 Undeclared structure/union/enum tag */`
	`struct st func2 ( int ) ;  /* F737 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。

28	【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(F750) */`
	`}`
	【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 ;`
	`}`

<table>
<tr><th>項番</th><th>注意事項</th></tr>
<tr><td>29</td><td>【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
<pre>
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 ( ) ;
}
</pre>【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。</td></tr>
<tr><td>30</td><td>【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
<pre>
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 */
}
</pre>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。</td></tr>
</table>

項番	注意事項

31	【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F750となります。
	例
	`struct Ss {`
	`    int  d1, d2 ;`
	`} ;`
	`union Au {`
	`    struct Ss t1;`
	`} u = { { 1, 2 } } ;   /* F750  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 } ;`

32	【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。
33	【スモール・モデル使用時の注意事項】 スモール・モデル（-MS）使用時には次のことに注意してください。 スモール・モデルでは，WHL, VVP, UUP, TDEのW, V, U, Tレジスタに0Hをスタートアップ時に設定し，その後はその値が保持されているものとしてコード生成が行われます（W, V, U, Tレジスタの値の設定は，CC78K4提供のスタートアップ・ルーチンで行われています）。 したがって，CC78K4提供のスタートアップ・ルーチンを使用せず，ユーザがアセンブラ・ソースを記述しリンクする場合は，最初にW, V, U, Tレジスタにユーザが0を設定し，その後のアセンブラ・ソース中でW, V, U, Tレジスタを破壊しないように注意してください。

項番	注意事項
34	【ミディアム・モデル使用時の注意事項】 ミディアム・モデル（-MM）使用時には次のことに注意してください。 （a）ミディアム・モデルは，コード1 Mバイト，データ64 Kバイトの配置が可能ですが，データはLOCATIONの値により，000000H-00FFFFHまたは0F0000H-0FFFFFHのいずれかの64 Kに配置されます。コード・セグメント@@CNSTS（LOCATION 0）および@@CNSTM（LOCATION 0F）を使用する場合は，必ずデータを配置する領域と同じ64 K内に配置してください。 （b）ミディアム・モデルでは，VVP, UUP, TDEのV, U, Tレジスタに LOCATION 0ならば0Hを，LOCATION 15ならば0FHをスタートアップ時に設定し，その後は，その値が保持されているものとしてコード生成が行われます（V, U, Tレジスタの値の設定は，CC78K4提供のスタートアップ・ルーチンで行われています）。したがって，CC78K4提供のスタートアップ・ルーチンを使用せず，ユーザがアセンブラ・ソースを記述しリンクする場合は，最初にV, U, Tレジスタにユーザが適切な値を設定し，その後のアセンブラ・ソース中でV, U, Tレジスタを破壊しないように注意してください。 （c）データのアドレスを参照する場合，データのアドレスの下位2バイトが0000Hの位置に配置された場合，NULLポインタとの比較の際に等価と判断されます（ミディアム・モデルは，データ・アドレスを下位2バイトしか持たないため）。
35	【saddr1/saddr2領域使用時の注意事項】 1つのファイルでextern sreg宣言した変数（saddr2領域使用）と同名の変数を，別ファイルで__sreg1（saddr1領域）宣言をしたり，その逆を行わないでください。 1つのファイルでextern bit/boolean/__boolean宣言した変数（saddr2領域使用）と同名の変数を，別ファイルで__boolean1（saddr1領域）宣言をしたり，その逆を行わないでください。 上記の宣言を行った場合，動作が保証されません。

付録C　CC78K4に関する制限事項一覧

ここでは，CC78K4の制限事項の詳細と，その回避方法について説明します。

項番	制限事項の概要
1	ブロック内でextern宣言された外部変数の初期化がエラーとならず，また，アセンブラ・ソース中のディバグ情報が不正となる制限
2	ブロック内でextern宣言された変数と同名の変数の結合が不正となる場合がある制限
3	関数プロトタイプ宣言または型修飾子const, volatileを使用した宣言に，typedefで定義された型（typedef名）を使用すると，typedefの展開が不正となり，エラーとなる制限
4	大きさが定義されていない多次元配列が，不正動作となる場合がある制限
5	引数を持つ関数のアドレスを返却する関数において，前記引数は参照できないが，参照時エラーとならず，不正なコードを出力する制限
6	signed型のビット・フィールドを，符号なしのビット・フィールドとして処理する制限
7	ラージ・モデルにおいて，1関数内のauto変数の総サイズが65535バイトを越えると，出力コードやディバグ情報が不正となる制限

C. 1　制限事項の詳細と回避方法

●制限事項1

ブロック内でextern宣言された外部変数の初期化がエラーにならず，また，アセンブラ・ソース中のディバグ情報が不正となる制限

【内　　容】

ブロック内でextern宣言された外部変数の初期化は，ANSI C言語仕様に合致しないためエラーとすべき記述ですが，エラーとなりません。コンパイラは，初期値ありの外部変数が定義されたものと解釈してコードを出力します。

コンパイラが出力したオブジェクト中のディバグ情報は正常ですが，アセンブラ・ソース中のディバグ情報が不正になります。

【再 現 例】

```
int i ;
void  f ( void ) {
              extern  int  i = 2;
}
```

【回 避 策】

ありません。

【発　　生】

Ver.1.00**以降**

●制限事項2

ブロック内でextern宣言された変数と同名の変数との結合が不正となる場合がある制限

【内　　容】

ブロック内でextern宣言された変数と同名の変数との結合が，以下の4つの条件のうち，いずれかに該当する場合に不正となります。

(1) ブロック内でextern宣言された変数と，以降のブロック外でstatic宣言された変数が同名である場合
この場合，エラーとならず，結合もしないため，この変数を参照すると不正なコードを出力します。

【再 現 例】

```
void  f ( void ) {
       extern  int  i ;
       i = 1 ;           /* 不正コード出力 */
}
static  int  i ;
```

(2) ブロック内でextern宣言された変数と，以降のブロック外でstatic宣言されない変数が同名である場合
この場合，結合せずに，不正なコードを出力します。

【再 現 例】

```
void  f ( void ) {
       extern  int  i ;
       i = 1 ;           /* 不正コード出力 */
}
int  i ;
```

(3) ブロック内でextern宣言された変数と，以前のブロック外でextern宣言されない変数が同名であり，さらにextern宣言された変数があるブロックを囲んでいるブロック中で宣言されている自動変数が同名である場合
この場合，ブロック外の変数とブロック内でextern宣言された変数は結合せず，不正なコードを出力します。

【再 現 例】

```
int  i  =  1 ;
void  f ( void ) {
     int  i ;
      {
         extern  int  i ;
         i = 1 ; /* 不正コード出力 */
     }
}
```

(4) ブロック内でextern宣言された変数と，他のブロック内でextern宣言された変数が同名である場合
この場合，結合せず，不正なコードを出力します。

【再 現 例】

```
void  f1( void ) {
       extern  int  i ;
       i = 2 ;
}
void  f2( void ) {
       extern  int  i ;
       i = 3 ;
}
```

【回 避 策】

ありません。

【発　　生】

Ver.1.00以降

●制限事項3

関数プロトタイプ宣言または型修飾子const，volatile**を使用した宣言に，**typedef**で定義された型（**typedef**名）を使用すると，**typedef**の展開が不正となり，エラーとなる制限**

【内　　容】

関数プロトタイプ宣言または型修飾子const，volatile**を使用した宣言に，**typedef**で定義された型（**typedef**名）を使用すると，**typedef**の展開が不正となり，エラーとなる場合があります。**

【再 現 例 1**】**

```
typedef  int  FTYPE(  );

FTYPE  func ;
int  func ( void ) ;          /* F713  Redefined  'func' */
```

【再 現 例 2**】**

```
typedef  int  VTYPE[2] ;
typedef  int  *VPTYPE[3] ;

const  VTYPE  *a ;
const  int  ( *a )[2] ;              /* F713  Redefined  'a' */
volatile  VPTYPE  b[2] ;
volatile  int  *volatile  b[2][3] ;   /* F713  Redefined  'b' */
```

【回 避 策】

ありません。

【発　　生】

Ver.1.00**以降**

●制限事項4

大きさが定義されていない多次元配列が，不正動作となる場合がある制限

【内　　容】

大きさが定義されていない多次元配列が，不正動作となる場合があります。

【再 現 例　1】

```
char  c [ ] [3] = { { 1 }, 2, 3, 4, 5 } ;              /* 不正コード */
```

【再 現 例　2】

```
char  c [ ] [2] [3] = { ″ab″, ″cd″, ″ef″ } ;           /* エラー(F756) */
```

【回 避 策】

多次元配列の大きさを定義してください。

【発　　生】

Ver.1.00**以降**

●制限事項5

引数を持つ関数のアドレスを返却する関数において，前記引数は参照できないが，参照時エラーとならず，不正なコードを出力する制限

【内　　容】

引数を持つ関数のアドレスを返却する関数において，前記引数は参照できないが，参照時エラーとならず，不正なコードを出力します。

【再 現 例　1】

```
char  *c ;
int  *i ;
void  ( * f1 ( int    * ) ) ( char * ) ;
void  ( * f2 ( void    ) ) ( char * ) ;
void  ( * f3 ( int    * ) ) ( void   ) ;

void  main ( ) {
    ( * f1 ( i ) ) ( c ) ;      /* 正しい記述     (W510) */
    ( * f1 ( i ) ) ( i ) ;      /* 間違った記述   */
    ( * f2 (  ) ) ( c ) ;       /* 正しい記述     (W509) */
    ( * f2 (  ) ) (    ) ;      /* 間違った記述   (W509) */
    ( * f3 ( i ) ) (    ) ;     /* 正しい記述     (W509) */
    ( * f3 ( i ) ) ( i ) ;      /* 間違った記述   */
}
```

正しい記述に対してワーニング・メッセージW509/W510を出力してしまいます。逆に，間違った記述に対してエラー・メッセージを出力しません。ただし，出力コードは正常です。

【再 現 例　2】

```
void  ( * f4 ( ) ) ( int p ) {
    p++ ;               /* 間違った記述 */
}
```

エラーにすべき記述に対してエラーを出力せず，不正コードを生成します。

【回 避 策】

ありません。

【発　　生】

Ver.1.00以降

●制限事項6

signed型のビット・フィールドを，符号なしのビット・フィールドとして処理する制限

【内　　容】

signed型のビット・フィールドを，符号なしのビット・フィールドとして処理します。

【回 避 策】

ありません。

【発　　生】

Ver.1.00以降

●制限事項7

ラージ・モデルにおいて，1関数内のauto変数の総サイズが 65536バイトを越えると，出力コードやディバグ情報が不正となる制限

【内　　容】

ラージ・モデルにおいて、1関数内のauto変数の総サイズが 65536バイトを越えると、出力コードやディバグ情報が不正となります。

【再 現 例】（-ML指定時）

```
void func(long a, int b){
    int i;
    char tab1[35000];
    char tab2[35000];

    i = b;
}
```

【回 避 策】

1関数内のauto変数の総サイズを 65535バイト以下にしてください。

【発　　生】

Ver.1.00以降

付録D　索　　引

アルファベットで始まる語句

50音で始まる語句の索引

〔メ　モ〕

【発　行】

NECエレクトロニクス株式会社

〒211-8668　神奈川県川崎市中原区下沼部1753

電話（代表）：044(435)5111

お問い合わせ先

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。

URL(アドレス)　http://www.necel.co.jp/

【営業関係，技術関係お問い合わせ先】

半導体ホットライン

（電話：午前 9:00～12:00，午後 1:00～5:00）

電　話　：044−435−9494

E-mail　：info@lsi.nec.co.jp

【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エレクトロニクス特約店へお申し付けください。