お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010 年 4 月 1 日
ルネサスエレクトロニクス株式会社

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

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

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

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

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

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

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

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

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

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

ユーザーズ・マニュアル

CA850 Ver.3.20

Cコンパイラ・パッケージ

操作編

対象デバイス
V850シリーズ

資料番号　U18512JJ1V0UM00（第1版）
発行年月　May 2007　CP(K)

〔メ　モ〕

目次要約

Windows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。

- 本資料に記載されている内容は2007年5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

対象デバイス　V850シリーズ Cコンパイラ・パッケージは，NECエレクトロニクス製RISCマイクロプロセッサV850シリーズ用のオブジェクト・コードを生成するためのコンパイラ・パッケージです。
このマニュアルは，CA850 Cコンパイラ・パッケージを対象としています。

対 象 者　このマニュアルは，V850シリーズ Cコンパイラ・パッケージを使用して，アプリケーション・システムを開発するユーザを対象としています。

目　　的　このマニュアルでは,各パッケージに含まれるCコンパイラ,アセンブラなどの各コマンドをWindows®上で操作する際の方法について説明しています。
なお，PM+（Windows版のみ）は，このCコンパイラ・パッケージに含まれて提供されていますが，統合開発としての操作方法については，PM+のユーザーズ・マニュアルを参照してください。
また，Windowsの操作方法については，Windowsに添付されている機能ガイドなどを参照してください。

構　　成　このマニュアルは，次の内容で構成されています。

- CA850の概要
- CA850をコマンド・ラインで利用する方法
- CA850をプロジェクト・マネージャから利用する方法
- 各コマンドの機能，オプションおよび出力メッセージ

パッケージに含まれるコマンド
Cコンパイラ（ca850）
アセンブラ（as850）
リンカ（ld850）
ROM化プロセッサ（romp850）
ヘキサ・コンバータ（hx850）
アーカイバ（ar850）
セクション・ファイル・ジェネレータ（sf850）
ダンプ・コマンド（dump850）
ディスアセンブラ（dis850）
クロス・レファレンス・ツール（cxref）
メモリ・レイアウト視覚化ツール（rammap）
スタック見積もりツール（stk850）

読み方の注意　・このマニュアルでは，Cコンパイラ・パッケージの各プログラム名を次のように記述します。

Cコンパイラ・パッケージ→CA850
アセンブラ→as850
Cコンパイラ→ca850

関連資料　**このマニュアルを使用する場合は，次の資料もあわせてご覧ください。**

関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。

あらかじめご了承ください。

○開発ツールに関する資料（ユーザーズ・マニュアル）

資　料　名		資料番号	
		和文	英文
CA850 Ver.3.20 **Cコンパイラ・パッケージ**	**操作編**	**このマニュアル**	U18512E
	C言語編	U18513J	U18513E
	アセンブリ言語編	U18514J	U18514E
	リンク・ディレクティブ編	U18515J	U18515E
PM+ Ver.6.30　**プロジェクト・マネージャ**		U18416J	U18416E
ID850 Ver.3.00　**統合ディバッガ**	**操作編**	U17358J	U17358E
ID850NW Ver.3.10　**統合ディバッガ**	**操作編**	U17369J	U17369E
ID850QB Ver.3.20　**統合ディバッガ**	**操作編**	U17964J	U17964E
SM+ **システム・シミュレータ**	**操作編**	U17246J	U17246E
	ユーザ・オープン・インタフェース編	U18212J	U18212E
SM850 Ver.2.50　**システム・シミュレータ**	**操作編**	U16218J	U16218E
SM850 Ver.2.00**以上**　**システム・シミュレータ**	**外部部品ユーザ・オープン・インタフェース仕様編**	U14873J	U14873E
RX850 Ver.3.20**以上**　**リアルタイム**OS	**基礎編**	U13430J	U13430E
	インストレーション編	U17419J	U17419E
	テクニカル編	U13431J	U13431E
	タスク・デバッガ編	U17420J	U17420E
RX850 Pro Ver.3.21　**リアルタイム**OS	**基礎編**	U18165J	U18165E
	内部構造編	U18164J	U18164E
	タスク・デバッガ編	U17422J	U17422E
RX850V4 Ver.4.22　**リアルタイム**OS	**機能編**	U16643J	U16643E
	内部構造編	U16644J	U16644E
	タスク・デバッガ編	U16811J	U16811E
RX-NET　**ネットワーク・ライブラリ**（TCP/IP）Ver.1.30		U15083J	−
RX-NET　**ネットワーク・ライブラリ**（PPP）Ver.1.30		U15303J	−
RX-NET　**ネットワーク・ライブラリ**（DNS）Ver.1.30		U15304J	−
RX-NET　**ネットワーク・ライブラリ**（DHCP）Ver.1.30		U15382J	−
RX-NET　**ネットワーク・ライブラリ**（SMTP）		U15505J	−
RX-NET　**ネットワーク・ライブラリ**（POP）		U15539J	−
RX-NET Ver.1.10 **ネットワーク・ライブラリ**（telnet）		U16085J	−
RX-NET Ver.1.00 **ネットワーク・ライブラリ**（FTP）		U15946J	−
RX-NET Ver.1.00 **ネットワーク・ライブラリ**（WebServer）		U16294J	−
AZ850 Ver.3.30　**システム・パフォーマンス・アナライザ**		U17423J	U17423E
AZ850V4 Ver.4.10　**システム・パフォーマンス・アナライザ**		U17093J	U17093E
TW850 Ver.2.00　**性能解析チューニング・ツール**		U17241J	U17241E

〔メ　モ〕

目次

図の目次

図番号　タイトル,ページ

表の目次

表番号　タイトル,ページ

第 1 章　概要

1.1　コンパイラ・パッケージ内容

V850 マイクロコントローラ用 C コンパイラ・パッケージには，次のものが含まれています。

1. C コンパイラ（ca850）
2. アセンブラ（as850）
3. リンカ（ld850）
4. ROM 化プロセッサ（romp850）
5. ヘキサ・コンバータ（hx850）
6. アーカイバ（ar850）
7. セクション・ファイル・ジェネレータ（sf850）
8. ダンプ・コマンド（dump850）
9. ディスアセンブラ（dis850）
10. クロス・リファレンス・ツール（cxref）
11. メモリ・レイアウト視覚化ツール（rammap）
12. スタック見積もりツール（stk850）
13. リンク・ディレクティブ・ジェネレータ（LDG）

なお，これらの起動は，次のどちらかの方法により行います。

(1)　統合開発環境 “ PM+ ” から起動

“ PM+ ” は，C コンパイラ・パッケージに含まれています。PM+ についての詳細は PM+ のユーザーズ・マニュアルを参照してください。

(2)　コマンド・ラインから起動

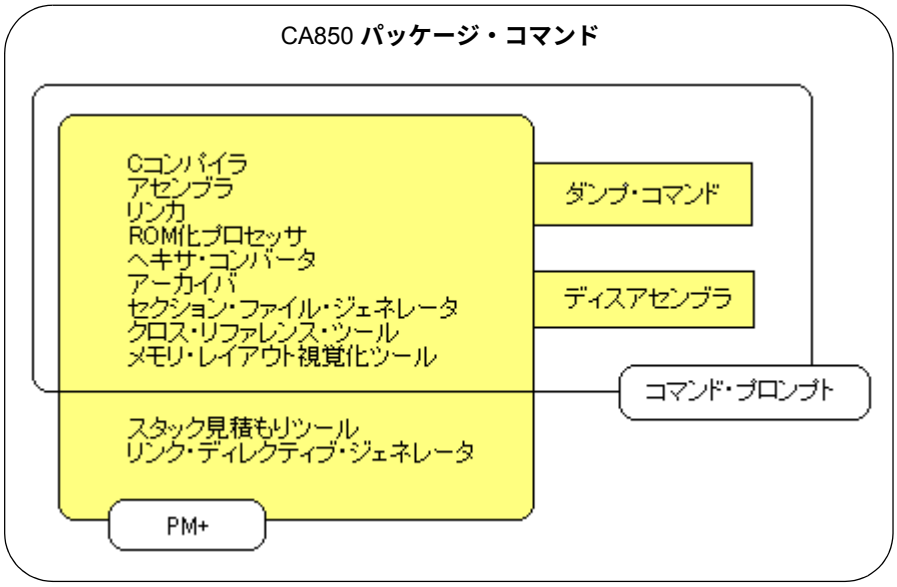
コマンド・プロンプトから，コマンドを入力して起動します。ロード・モジュールをバッチファイルやメイクファイルを使用して生成するときは，コマンド入力形式で記述します。

また，8 ～ 11 のユーティリティもコマンド入力によって起動することができます。

コマンドの入力方法についての詳細は，各ツールの「操作方法」の項を参照してください。

PM+ を使用する場合のパッケージ構成は，次のようになります。

図 1 － 1 パッケージ構成概要図



PM+ から上記の “ CA850 パッケージ・コマンド ” に含まれる “ C コンパイラ ”, “ アセンブラ ”, および “ リンカ ” などのオプション指定をウインドウ上で行い，それらのコマンドを PM+ が自動的に起動してロード・モジュールを作成します。

なお，CA850 を使用するには，デバイス情報を持つ “ デバイス・ファイル ” が別途必要になります。ご使用のデバイスに応じたデバイス・ファイルを入手してください。

1.2 動作環境

C コンパイラ・パッケージを使用するには，次の環境が必要となります。

(1) ホスト・マシン

CPU ： Pentium™ 400MHz 以上

メモリ ： 128M バイト以上

OS ： Windows® 2000，Windows XP Professional，Windows XP Home Edition

注 いずれの OS の場合も，最新の Service Pack がインストールされている必要があります。

(2) 対応ツール

- 統合デバッガ
  ID850（Ver.3.10 以降），ID850NW（Ver.3.10 以降），ID850QB（Ver.3.30 以降）
- システム・シミュレータ
  SM850（Ver.3.00 以降），SM+ for V850（Ver.2.00 以降）
- 性能解析チューニング・ツール
  TW850（Ver.2.10 以降）
- リアルタイム OS
  RX850（Ver.3.20 以降），RX850 Pro（Ver.3.20 以降）
- タスク・デバッガ
  RD850（Ver.3.20 以降），RD850 Pro（Ver.3.20 以降）
- システム・パフォーマンス・アナライザ
  AZ850（Ver.3.30 以降）
- デバイス・ファイル・インストーラ
  DFINST（Ver.3.10 以降）

注意 CA850 を使用するには，デバイス情報を持つデバイス・ファイルが別途必要になります。
ご使用のデバイスに応じたデバイス・ファイルを，次のバージョン・アップ・サービスから入手してください。

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

第 2 章　インストレーション

この章では，CA850 のインストール／アンインストールについて説明します。

2.1　インストール

CA850 を使用するには，CA850 とデバイス・ファイルのインストールが必要になります。また，PM+ を使用する場合では，この際に PM+ のインストールを行います。

CA850 のインストールは，次の手順で行います。
なお，CA850 の提供媒体は，CD-ROM（１枚）です。

(1)　Windows を起動します。

(2)　ホスト・マシンの CD ドライブに CD-ROM を挿入します。
その後，自動的にセットアップ・プログラムが起動されます。もし，自動的にセットアップ・プログラムが起動されない場合には，Windows のエクスプローラを起動し，CD ドライブ中の “ Install.exe ” をダブル・クリックしてください。

(3)　インストールするツール（CA850，PM+，STK850，LDG など）を選択し，インストール先フォルダを指定して [インストール] ボタンをクリックします。

(4)　以降，画面に表示されるメッセージに従ってインストールを実行してください。

備考　デバイス・ファイルは，この際にインストールされるデバイス・ファイル・インストーラ（DFINST）に従ってインストールしてください。

2.2 フォルダ構成

CA850 をインストールした結果，提供媒体から読み込まれた各種ファイルのフォルダ構成を，図 2 － 1 に示します。

なお，CA850 の標準フォルダ（デフォルト・インストール）は，次のとおりです。

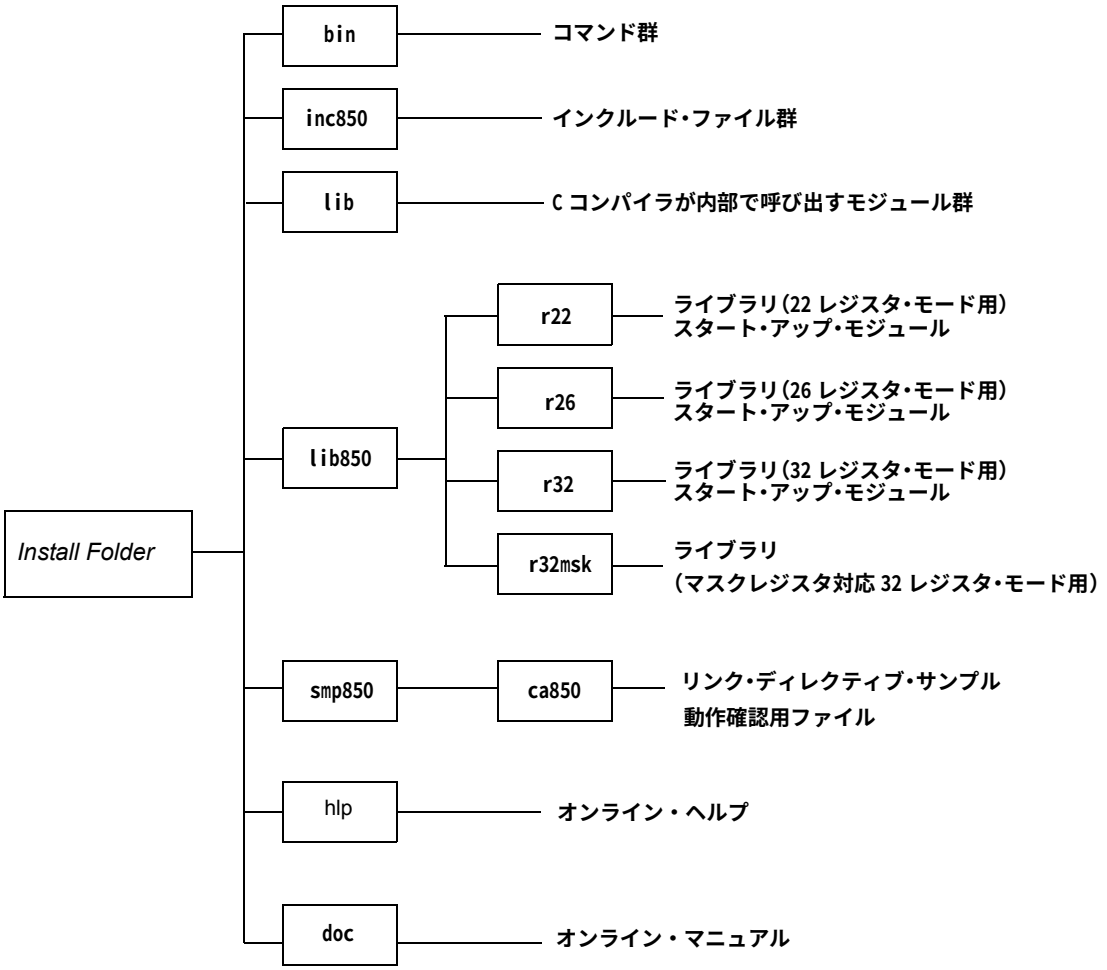
```
C:\Program Files\NEC Electronics Tools\CA850\Vx.xx
```

CA850 では，図 2 － 1 に示したフォルダ lib850 をライブラリに対する標準フォルダと呼び，フォルダ inc850 をインクルード・ファイルに対する標準フォルダと呼びます。

なお，デバイス・ファイルの標準フォルダ（デフォルト・インストール）は，次のとおりです。

```
C:\Program Files\NEC Electronics Tools\DEV
```

図 2 － 1 フォルダ構成



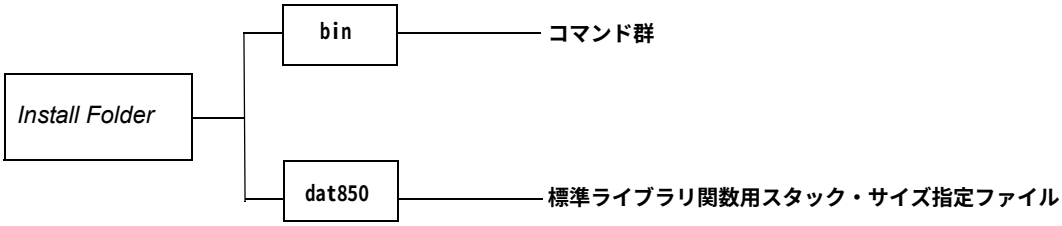
2.2.1 スタック見積もりツールのフォルダ構成

スタック見積もりツールをインストールした場合，提供媒体から読み込まれる各種ファイルのフォルダ構成を，図２－２に示します。

なお，スタック見積もりツールの標準フォルダ（デフォルト・インストール）は次のとおりです。

```
C:\Program Files\NEC Electronics Tools\STK850\Vx.xx
```

図２－２ スタック見積もりツールのフォルダ構成



2.3 アンインストール

ここでは，CA850 のアンインストール方法について説明します。

(1) Windows を起動します。

(2) Windows のコントロール・パネルの “ プログラムの追加と削除 ”（WindowsXP 以外の場合は，“ プログラムの追加と削除 ”）を起動します。

(3) 表示される一覧から，次の項目を選択することにより行います。
- NEC EL CA850 Vx.xx
- NEC EL CA850 Vx.xx ドキュメント一式

備考 他のツール（PM+，STK850，LDG など）のアンインストールも，上記の方法と同様に行います。

第 3 章　C コンパイラ

この章では，C コンパイラ（ca850）の概要，操作方法について説明します。

3.1　動作の流れ

ca850 は，C 言語ソース・ファイルに記述された C 言語のソース・プログラムから，リロケータブルなオブジェクト・ファイルや，ターゲット・システムで実行可能なオブジェクト・ファイルを生成します。

つまり，ca850 は，パッケージに含まれているモジュールのドライバとしての役目をし，マクロ展開やコメント処理，中間言語ファイルのマージ，最適化，アセンブリ言語ソース・プログラムの生成から機械語命令への変換，オブジェクト・ファイルのリンクといった操作ができます。

ca850 は，次の順序で処理を行います。

ただし，図 3 － 1 に示すように，最適化レベル指定によって処理の流れが若干異なります。

(1)　フロントエンド（cafe）が，C 言語ソース・プログラムに対するマクロ展開，コメントの処理を行ったのち，中間言語 OPTIC プログラムに変換します。

(2)　プリオプティマイザ（popt）が，中間言語 OPTIC プログラム中の関数の並び替えを行います。

また，コマンド・ラインからの起動では，マージ・オプション（-Om）を指定した場合，複数の中間言語 OPTIC プログラムを 1 つにマージします。

なお，より高度な最適化（実行速度優先）を指定した場合，さらに，中間言語 OPTIC プログラム中の関数のインライン展開を行います。

(3)　広域最適化部（opt）が，中間言語 OPTIC プログラムを最適化します。

(4)　コード生成部（cgen）が，中間言語 OPTIC プログラムをアセンブリ言語ソース・プログラムに変換します。

(5)　機種依存最適化部（impr）が，アセンブリ言語ソース・プログラムを最適化します。

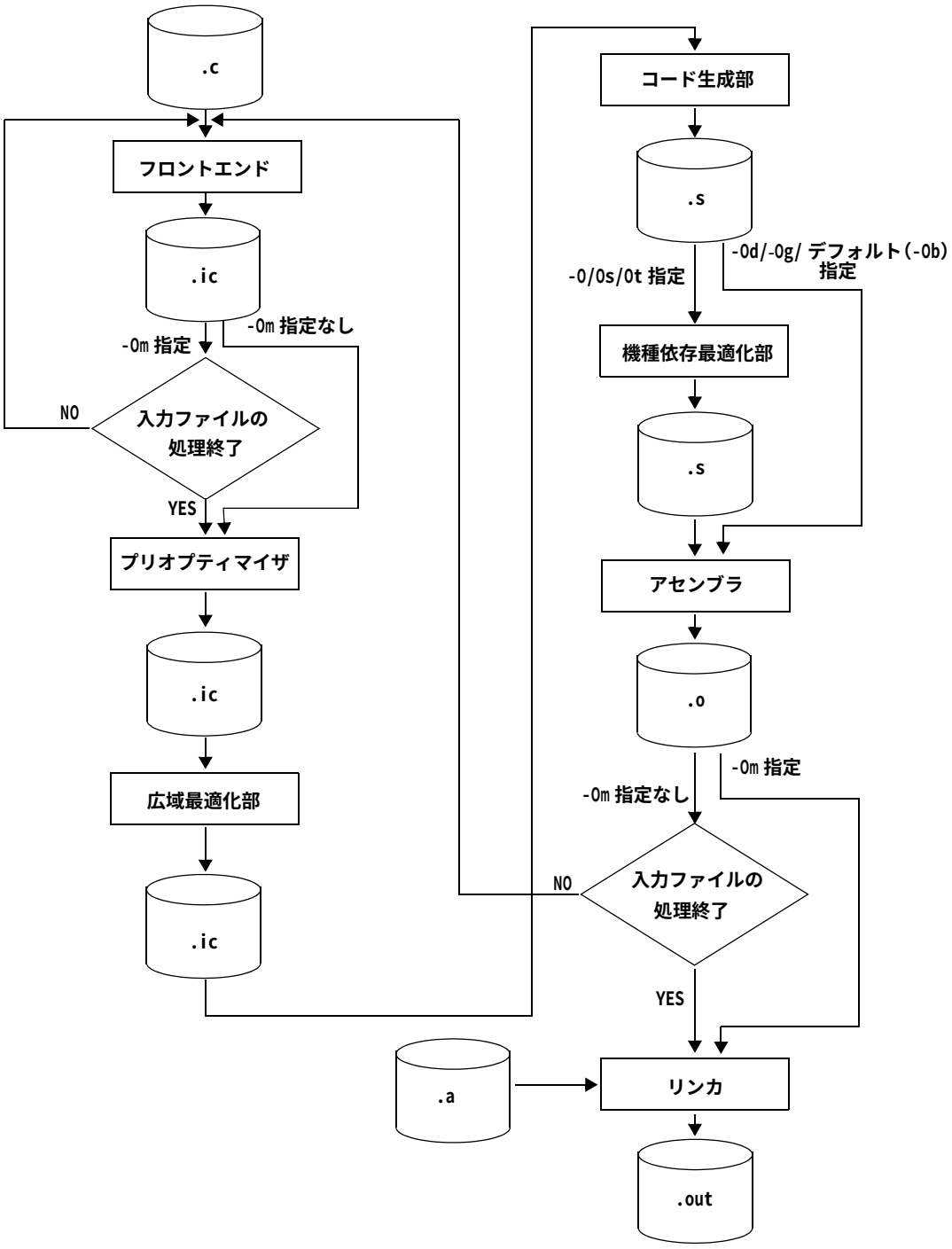
(6)　アセンブラ（as850）が，アセンブリ言語ソース・プログラムを機械語命令に変換し，リロケータブルなオブジェクト・ファイルを生成します。

(7)　リンカ（ld850）が，リロケータブルなオブジェクト・ファイルをリンクし，実行可能なオブジェクト・ファイルを生成します。

広域最適化部，および機種依存最適化部は，最適化オプションが指定された場合にのみ起動されます。

なお，(1) フロントエンド（cafe）～ (5) 機種依存最適化部（impr）のモジュールは，ca850 から起動されることを想定しています。したがって，これらを単独で起動した場合，動作は保証されません。

図 3 － 1 ca850 における動作の流れ



3.2 入出力ファイル

ca850 では，次のファイルを入力ファイル，または出力ファイルとして指定できます。

file.c	C 言語ソース・ファイル（.c ファイルと呼ばれる）
file.ic	中間言語ファイル（.ic ファイルと呼ばれる）
file.s	アセンブリ言語ソース・ファイル（.s ファイルと呼ばれる）
file.o	オブジェクト・ファイル（.o ファイルと呼ばれる）
file.a	アーカイブ・ファイル（.a ファイルと呼ばれる）

- .s ファイルは，そのまま as850 に渡されます（アセンブリ言語で直接記述されたソース・プログラムに対しては，機種依存最適化部を起動しません）。
- .a ファイルや .o ファイルなど，.c ファイル，.ic ファイル，および .s ファイル以外のファイルは，すべてそのまま ld850 に渡されます。

注 入力ファイル名は Windows で認められるものであれば指定できますが，‘ @ ’ はコマンド・オプションと判断されるため ‘ @ ’ をファイル名の先頭に使用できません。
また，ファイルの漢字コードが EUC の場合，ファイル，フォルダ，フォルダ名に日本語は使用できません。

3.3 実行オブジェクト

ca850 は，as850 や ld850 も起動するため，C 言語ソース・ファイルを読み込んで，実行可能なオブジェクト・ファイルの生成まで一度に行うことができます。

また，コマンド・ラインからのオプション（-S）の指定や PM+ の 1 ソース・コンパイル指定により，as850 や ld850 を起動する手前で処理を止め，コンパイラのコード出力やリロケータブルなオブジェクト・ファイルを生成することもできます（操作方法の詳細は「3.4 操作方法」を参照）。

各コマンドのコマンド・ラインからの起動例を，次に示します（オプションの詳細は「3.5 オプションの種類と機能」を参照）。

(1) ca850 からすべて行う場合

```
> ca850 -cpu 3201 file.c obj.o
```

デバイスに“ -cpu 3201 ”（V850ES/SA2）を指定し，*file*.c と *obj*.o を読み込み，実行可能なオブジェクト・ファイル a.out を作成します。このとき，スタート・アップ・モジュールとして crtE.o をリンクし，標準ライブラリ libc.a と libm.a を参照します。

```
> ca850 -cpu 3201 -R org_crt.o file.c obj.o
```

file.c と *obj*.o を読み込み，実行可能なオブジェクト・ファイル a.out を作成します。このとき，スタート・アップ・モジュールとして *org_crt*.o をリンクし，標準ライブラリ libc.a と libm.a を参照します。

(2) ca850 から as850 までを起動し，ld850 は単独で起動する場合

```
> ca850 -cpu 3201 -c file.c asm.s
```

file.c，*asm.s* を読み込み，リロケータブルなオブジェクト・ファイル *file.o*，*asm.o* を作成します。

```
> ld850 -cpu 3201 org_crt.o file.o asm.o obj.o -lc
```

org_crt.o，*file*.o，*asm*.o，*obj*.o をリンクし，実行可能なオブジェクト・ファイル a.out を作成します。このとき標準ライブラリ libc.a を参照します。

(3) ca850，as850，ld850 ともに単独で起動する場合

```
> ca850 -cpu 3201 -c file.c
```

file.c を読み込み，リロケータブルなオブジェクト・ファイル *file*.o を作成します。

```
> as850 -cpu 3201 asm.s
```

asm.s を読み込み，リロケータブルなオブジェクト・ファイル *asm*.o を作成します。

```
> ld850 org_crt.o file.o asm.o -lc
```

org_crt.o，*file*.o，*asm*.o をリンクし，実行可能なオブジェクト・ファイル a.out を作成します。このとき標準ライブラリ libc.a を参照します。

3.4 操作方法

この節では，ca850 の操作方法について説明します。

3.4.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
ca850 [ オプション ] ... ファイル名 [ ファイル名，またはオプション ]...
        [ ]：  [ ] 内は省略できます。
        ...：  直前の [ ] 内のパターンの繰り返しができます。
```

3.4.2 PM+ による方法

C 言語ソース・ファイルに対する [コンパイラオプションの設定] ダイアログは，PM+ でプロジェクトを設定後，次のいずれかの操作で表示します。

- 対象プロジェクト全体の C 言語ソース・ファイルに設定する場合

(1) [ツール] メニュー→ [コンパイラオプションの設定] を選択

- 個別の C 言語ソース・ファイルに設定する場合

(1) PM+ のプロジェクト・ウインドウ内で，オプションを設定したいソース・ファイル名を選択

(2) 右クリック・メニュー→ [個別コンパイラオプションの設定] を選択

設定は次のようになります。

- タイトル選択状態：全体オプション設定
- “ ソース・ファイル ” フォルダを選択状態：全体オプション設定
- 拡張子 “ .c ” のソース・ファイル名を選択状態：個別オプション設定

3.5 オプションの種類と機能

この節では，ca850 のオプションを示します。

なお，コマンド・ラインからの起動において ca850 のオプションにないオプションが与えられた場合，それらのオプションは，ld850 のオプションとみなし，そのまま ld850 に渡されます。

ca850 のオプションには，PM+ 上では設定できないオプションが存在します。このようなオプション指定が必要な場合は，コマンド・ラインから ca850 を起動してください。

〔オプション説明でのマーク〕

【V850E2】	V850E2 コア専用のオプション
【V850E】	V850Ex コア専用のオプション
【PM+】	PM+ で指定項目が存在するオプション
【78K 互換】	78K マイクロコントローラ C コンパイラ CC78K*x* 互換オプション

3.5.1 バージョン／ヘルプ表示／動作状態

次にバージョン／ヘルプ表示／動作状態オプションを示します。

-v

ca850 のバージョン情報を標準エラー出力に出力します。コンパイルは行いません。

-help

オプションの説明を標準エラー出力に出力します。

-v

【PM+】

ca850 の実行状況の詳細を標準エラー出力に出力します。

3.5.2 出力ファイル指定

次に出力ファイル指定オプションを示します。

-Fic[=outfile]

コンパイル途中に生成される OPTIC ファイルの保存先を指定します。

(a) *outfile* にファイル名を指定した場合

指定したファイル名でカレント・フォルダに *outfile* を保存します。

なお，*outfile* の拡張子は，.ic に限られます。

(b) *outfile* にフォルダを指定した場合

指定フォルダに .c を .ic で置き換えたファイル名で保存します。

(c) =*outfile* を省略した場合

カレント・フォルダに .c を .ic で置き換えたファイル名で保存します。

(d) 出力ファイルが複数の場合

outfile に指定したフォルダを作成し，.c を .ic で置き換えたそれぞれのファイル名で保存します。

-Fo[=*outfile*]

コンパイル途中に生成されるオブジェクト・ファイルの保存先を指定します。

(a) *outfile* にファイル名を指定した場合

指定ファイル名でカレント・フォルダに *outfile* を保存します。

(b) *outfile* にフォルダを指定した場合

指定フォルダに .c，または .s を .o で置き換えたファイル名で保存します。

(c) =*outfile* を省略した場合

カレント・フォルダに .c，または .s を .o で置き換えたファイル名で保存します。

(d) 出力ファイルが複数の場合

outfile に指定したフォルダを作成し，.c，または .s を .o で置き換えたそれぞれのファイル名で保存します。

-Fs[=*outfile*]

【PM+】

コンパイル途中に生成されるアセンブリ言語ファイルの保存先を指定します。

(a) *outfile* にファイル名を指定した場合

指定ファイル名でカレント・フォルダに *outfile* を保存します。

(b) *outfile* にフォルダを指定した場合

指定フォルダに .c を .s で置き換えたファイル名で保存します。

(c) =*outfile* を省略した場合

カレント・フォルダに .c を .s で置き換えたファイル名で保存します。

(d) 出力ファイルが複数の場合（PM+ では指定できません

outfile に指定したフォルダを作成し，.c を .s で置き換えたそれぞれのファイル名で保存します。

-Fv[=*outfile*]

【PM+】

コンパイル途中に生成されるアセンブル・リストの保存先を指定します。

(a) ***outfile* にファイル名を指定した場合**

指定ファイル名でカレント・フォルダに *outfile* を保存します。

(b) ***outfile* にフォルダを指定した場合**

指定フォルダに .c **を** .v **で置き換えたファイル名で保存します。**

(c) **=*outfile* を省略した場合**

カレント・フォルダに .c **を** .v **で置き換えたファイル名で保存します。**

(d) **出力ファイルが複数の場合（**PM+ **では指定できません**

***outfile* に指定したフォルダを作成し，**.c **を** .v **で置き換えたそれぞれのファイル名で保存します。**

このオプションと -a **オプションを指定しない場合は，アセンブル・リストは生成されません。**

-o *outfile*

出力ファイルを *outfile* で指定します。コンパイラ制御オプション -S，-c，-m **を指定しコンパイルを途中で止めた場合にもこのオプションは有効となります。**

(a) -S **オプションと同時に指定した場合**

***outfile* は，アセンブリ言語ファイル（**.s**）の指定となります。**

(b) -c **オプションと同時に指定した場合**

***outfile* は，リロケータブル・オブジェクト・ファイル（**.o**）の指定となります。**

(c) -m **オプションと同時に指定した場合**

***outfile* は，フロントエンド出力ファイル（**.ic**）の指定となります。**

(d) **上記以外の場合**

***outfile* は，実行可能なオブジェクト・ファイル（**.out**）の指定となります。デフォルトは** a.out **です。**

(e) **出力ファイルが複数の場合**

エラーとなります。

-temp=*dir*

【PM+】

内部的に用いるテンポラリ・ファイルを生成する作業用フォルダを指定します。このオプションを省略した場合，テンポラリ・ファイルは，環境変数 TEMP **で指定されたフォルダ，またはカレント・ドライブのルート・フォルダに生成されます。**

ハード・ディスクの容量不足などで，テンポラリ・ファイルを生成することができないためのエラーが発生した場合，このオプションで回避できます。

3.5.3 ソース・デバッガ制御

次にソース・デバッガ制御オプションを示します。

-Xno_word_bitop

【PM+】

ld.w ／ ld.h，st.w ／ st.h 命令を 1 ビット操作命令（set1，clr1，tst1，not1）へ置き換える動作を禁止します。デバッグ時に変数の read ／ write イベントを設定する場合，1 ビット操作命令に置き換わっていると，イベントが発生しないことがあります。この場合，このオプションを指定すると，ld.w ／ ld.h，st.w ／ st.h 命令のままとなり，デバッグがしやすくなります。

-g

【PM+】

ソース・デバッガ用のシンボル情報を出力します。つまり，このオプションの指定により，C 言語ソース・レベルでデバッグが可能となります。

ca850 から as850 を起動する場合，このオプションの指定により，as850 の -g も指定したものとみなされます。これによりデバッガでアセンブリ言語ソース・レベルでデバッグが可能となります。

3.5.4 コンパイル・ドライバの制御

次にコンパイル・ドライバの制御オプションを示します。

(1) デバイス指定関係オプション

-X256M

【V850E】【PM+】

メモリ空間を 256M バイトとして扱います。このオプションを指定しない場合はメモリ空間を 64M バイトとして扱い，アドレス解決します。このオプションは使用するチップセットにあわせて設定してください。V850Ex コアでは，物理アドレス空間が 256M バイト持つ場合が多く，64M を越えた 256M までの空間を使用するアプリケーションを作成する場合，このオプションを指定してください。

-Xbpc=*num*

【PM+】

プログラマブル周辺 I/O レジスタの上位アドレスを設定します。*num* には，BPC レジスタの最上位ビットを除いたアドレス部分のみを指定してください。ターゲット・デバイスがプログラマブル周辺 I/O レジスタ機能を持ち（V850E/IA1 など），変更可能なアドレス部分（=BPC レジスタに設定する値）を設定したい場合，アプリケーションのコンパイル（アセンブル）時に，値を確定させる必要があります。そこでこのオプションを指定すると，指定した値を使用してコンパイル（アセンブル）します。このオプション指定時には必ず値を指定してください。値には 2 進数，8 進数，10 進数，16 進数を使用できます。不正な値を指定した場合，および BPC レジスタに設定可能な範囲を越える値を指定した場合，警告メッセージが出力され，このオプションは無視されます。

例

```
-Xbpc=0x1234
```

上記の場合，ターゲット・デバイスが V850E/IA1 の場合，プログラマブル周辺 I/O レジスタ領域の先頭アドレスは，この値を 14 ビット左シフトした 0x48d0000 として扱われます。

設定する値はアプリケーション全体で 1 つです。ファイルごとのオプション設定で“ -Xbpc ”や“ -bpc ”を指定する場合，ファイル間で同じ値にしてください。ただし，プログラマブル周辺 I/O レジスタを使用しないファイルに対しては，このオプションを指定する必要はありません。

プログラマブル周辺 I/O レジスタ機能を持たないターゲット･デバイスの場合,および V850 コア／ V850Ex コア／ V850E2 コア共通としてアセンブルする場合，このオプションを指定すると，警告メッセージが出力され，このオプションは無視されます。

なお，このオプションは，プログラマブル周辺 I/O レジスタのアドレスをコンパイル（アセンブル）時に確定するためのものであり，BPC レジスタに実際に値を反映させるものではありません。動作のためには，別途，スタート・アップ・モジュールなどで BPC レジスタに値を設定する必要があります。

“ CA850 ユーザーズ・マニュアル C 言語編 ”にスタート・アップ・ルーチンのサンプルが載っていますので，そちらを参照してください。また，パッケージに含まれるスタート・アップ・モジュールにも，サンプルが載っています（コメントアウトしてあります）。

例

V850E/IA1 で，プログラマブル周辺 I/O レジスタの先頭アドレスの変更可能部分を “ 0x1234 ” とし，この機能の使用を許可するフラグ “ 0x8000 ” を設定する場合，次の記述をスタート・アップ・モジュールに記述します。

```
mov    0x9234,r10    -- 0x1234 | 0x8000 = 0x9234
st.h   r10, BPC
```

as850 では，このオプションを指定するか，または省略していても，実際にはプログラマブル周辺 I/O レジスタを参照していた場合，予約セクションである .bpc セクションを出力します。

このセクションは，リンク時のチェックのために使用されます。.bpc セクションは，情報用の特殊な予約セクションであり，メモリにロードされることはありません。したがって，通常のセクションのように，リンク・ディレクティブに記述する必要はありません。

-cn

生成するオブジェクトに V850 コア共通のマジックナンバを埋め込みます。

-cnv850e

【V850E】

生成するオブジェクトに V850Ex コア共通のマジックナンバを埋め込みます

-cnv850e2

【V850E2】

生成するオブジェクトに V850E2 コア共通のマジックナンバを埋め込みます

-cpu *devicename*

ターゲット・デバイスを指定します[注]。PM+ 使用の場合 [プロジェクトの設定] のプロジェクト情報のタブ選択時に行うデバイス指定に相当します。

このオプションを省略し，-cn ／ -cnv850e ／ -cnv850e2 オプション，または #pragma 指令による指定もない場合は，コンパイルは中止されます。

注 “ #pragma cpu *devicename* ” と同じ機能です。

-cpu オプション指定と #pragma 命令による指定の両方があり，指定の内容が異なっている場合，-cpu オプションによる指定内容が優先されます。

-devpath=*dir*

デバイス・ファイルを，フォルダ *dir* から検索します。このオプションを省略した場合，標準フォルダを検索します。

なお，PM+ を使用したときは，デバイス・ファイルのインストール先が自動的に設定されるため，このオプションを意識する必要はありません。

(2) コンパイラ制御指定オプション

`-S`

as850 以降のモジュールを実行せず，生成されたアセンブリ言語ソース・プログラムを出力します。

出力されるファイル名は，.c を .s で置き換えたものになります。-o オプションを利用することで，出力ファイル名を指定できます（-o オプションの説明参照）。また，-Fs オプションでも出力ファイル名を指定できます。

このオプションを省略した場合，as850 以降のフェーズも実行されます。

`-a`

アセンブル・リストを出力します。ファイル名は .c を .v に置き換えたものになります（「4.6 アセンブル・リスト」参照）。

なお，最適化オプションで，“ 標準最適化（-Og）” を指定した場合，アセンブラによる最適化のための命令並び替えにより，アセンブル・リストの出力が，一部不正確になる場合があります。

PM+ でこのオプション（-Fv オプション）を使用すると，アセンブル・リストのファイル名を指定することができます。

`-c`

ld850 まで起動せず，オブジェクト・ファイルの出力まで行います。ファイル名は，.c，または .s を .o で置き換えたものになります。-o オプションを利用することで，出力ファイル名を指定できます（-o オプションの説明参照）。また，-Fo オプションでも出力ファイル名を指定できます。

このオプションを省略した場合，ld850 まで起動します。

なお，PM+ を使用する場合は，すべてのコンパイルに対し自動的にこのオプションが指定されます。

`-m`

フロントエンドのみ実行し，.ic ファイルを生成して終了します。

このオプションを省略した場合，フロントエンド以降のモジュールも実行します。

(3) ROM 化制御オプション

`-Xr`

【PM+】

ROM 化用のオブジェクトを作成する場合に必要なオプションで，リンク処理に続いて ROM 化プロセッサを起動します。生成されるオブジェクト・ファイル（デフォルト名は romp.out）は，ROM 化情報を持つファイルとなります。

コンパイラが行う処理は，次のようになります。

(a) 先頭が “ _rcopy ”で始まる関数の第一引数のラベルが，オブジェクト内の .text セクションの終端を越える最初の（4 バイトの整列条件で整列された）アドレスを指すようにする。

(b) これにより，rompsec セクション用の領域確保コード（デフォルト名は rompcrt.o），および libr.a ファイルを，リンカ（ld850）でリンクするように指示。

ROM 化オブジェクトの作成方法の詳細は「6.4 ROM 化用オブジェクトの作成」を参照してください。

(4) プリプロセッサ処理設定オプション

-C

【PM+】

C 言語ソース・プログラムの前処理の出力に，ソース・プログラムのコメントも含めます。-E，または -P オプション指定時のみ有効です。

-D*name*[=*def*]

【PM+】

C 言語ソース・プログラムの前に #define *name def* が記述されたものとみなします。=*def* の指定を省略した場合，*def* を 1 とみなします。256 個まで指定できます。

-E

C 言語ソース・プログラムの前処理のみ実行し，結果を標準出力に出力します。結果には，ソース・プログラムの行番号表示と，ファイル名表示を含みます。

-I*dir*

【PM+】

C 言語ソース・プログラムのヘッダ・ファイルをフォルダ *dir*，標準フォルダの順で検索します。100 個まで指定できます。

このオプションを省略した場合，標準フォルダのみ検索します。標準フォルダは，インストール・フォルダ \inc850 フォルダです。また，#include " ヘッダ・ファイル名 " と記述された場合は，ソース・ファイルのあるフォルダを最初に検索します。

-P

C 言語ソース・プログラムに対し前処理のみ実行して，結果を C 言語ソース・ファイル名の .c を .i に置き換えた名前のファイルに出力します。

ソース・プログラムの行番号表示やファイル名表示は出力しません。

-U*name*

【PM+】

C 言語ソース・プログラムの前に #undef *name* が記述されたものとみなします。256 個まで指定できます。

-Wa,-D*name*[=*num*]

【PM+】

アセンブリ言語ソースの前に .set *name*, *num* が記述されたものとみなします。“ =*num* ” の指定を省略した場合，*num* を 1 とみなします。

-Wa,-I,*dir*

【PM+】

アセンブリ言語ソース・ファイルのヘッダ・ファイルをフォルダ *dir*，標準フォルダの順で検索します。このオプションを省略した場合，標準フォルダのみ検索します。

-Xcxxcom

【PM+】

通常のコメントのほかに，“ // ” から行末までをコメント（C++ のコメント・スタイル）として扱います。

-Xd

自動変数ではない変数のアドレス，または関数のアドレスを用いたポインタ型外部変数の初期化に対し，警告メッセージを出力します。

-Xm*num*

【PM+】

マクロ識別子数の上限を指定します。*num* には，32,767 までの 10 進数を指定します。このオプションを省略した場合，2,047 とします。

なお，このオプションはプリプロセッサで使用するバッファのサイズを大きくするものです。

ただし，これによって何文字分のバッファが確保されるのかという具体的な数値を出すことはできません。

-t

【PM+】

トライグラフ系列の置換を行います。ANSI 規格で規定された，単一文字に置換される 3 文字（トライグラフ）系列です。詳細は ANSI 規格に関する文献を参照してください。

(5) コンパイル時のメモリ節約オプション

-Wp,-D

【PM+】

コンパイル時のプリオプティマイザのメモリ使用量を減らします。マシンのメモリが不足しコンパイルが正常に終了しないときにこのオプションを指定します。このオプションを指定するとコンパイル速度は低下します。

-Wi,-D

【PM+】

コンパイル時の機種依存最適化のメモリ使用量を減らします。マシンのメモリが不足しコンパイルが正常に終了しないときにこのオプションを指定します。このオプションを指定するとコンパイル速度は低下します。

(6) エラー出力指定オプション

+err_file=*file*

エラー・メッセージをファイル *file* に追加保存します。

なお，PM+ では，[コンパイラ共通オプションの設定] ダイアログの [ファイル] タブにおける“ エラーファイル ”にファイル名を指定するとこのオプションを指定したことと同じになります。

-err_file=*file*

エラー・メッセージをファイル *file* に上書き保存します。

-err_limit=*num*

【PM+】

エラー・メッセージの最大出力数 *num* を指定します。*num* には，15 から 50 までの 10 進数を指定します。このオプションを省略した場合，15 とします。

(7) 拡張機能指定オプション

-cc78k

【78K 互換】【PM+】

78K マイクロコントローラ C コンパイラ CC78K*x* 互換の拡張機能を有効にします。

3.5.5 最適化

“最適化”は，アプリケーションの実行速度を向上したり，使用 ROM 容量を小さくしたりする処理です。最適化のかかり方は，最適化のレベルによって異なります。最適化レベルの高い最適化を選択した場合，コンパイル速度が低下したり，削除／変更される C 言語ソース行や変数のレジスタ化の確率が高くなります。後者の場合，デバッガにおいてブレークポイントが設定できなくなる現象などが発生する可能性があり，デバッグ効率が悪くなることがあります。

最適化についての詳細は「3.7.3 効率的な最適化の仕方」を参照してください。

(1) 最適化オプション

`-Od`

【PM+】

デバッグ優先オプションです。ROM 容量や実行速度に着目せず，論理デバッグに着眼したコードを生成します。CA850 Ver.2.41 以前のデフォルト最適化に相当する機能です。

`-Ob`

【PM+】

デフォルト・オプションです。論理デバッグに着眼したコードを生成します。論理デバッグに影響のない範囲で最適化を行います。

`-Og`

【PM+】

標準最適化オプションです。適度な最適化を行います。ほとんどの場合で C 言語ソース・デバッグが可能となる最適化を行います。外部変数をレジスタに割り当てることから，実行速度，コード・サイズともにデフォルト・オプションより改善されます。

`-O`

【PM+】

高度な最適化オプションです。ROM 容量に着目した最適化を行います。

`-Os`

【PM+】

より高度な最適化（オブジェクト・サイズ優先）オプションです。ROM 容量を最も重視した最大限の最適化を行います。

`-Ot`

【PM+】

より高度な最適化（実行速度優先）オプションです。ROM 容量よりも実行速度を最も重視した最大限の最適化を行います。

(2) ターゲット・コード最適化オプション

-Wi,-O4

【PM+】

データ・フロー解析を厳密に行い，最も強い最適化を行います。最適化オプション -O ／ -Os ／ -Ot を指定した上で，さらに強い最適化を行いたい場合に指定してください。この最適化では具体的に次のことを行います。

- 分岐命令をまたいだレジスタの最適化
- 絶対値演算の最適化
- 分岐命令をまたいだ cmp 命令の最適化
- 分岐命令をまたいだ復帰命令の最適化

なお，ソースによっては -Os ／ -Ot の最適化結果と同じになることがあります。また，コンパイル時間は，-Os ／ -Ot 指定時より遅くなります。

-Wi,-P

【PM+】

分岐先ラベルを整列する最適化を抑止します。これにより，実行コードのサイズを小さくすることができます。このオプションは，より高度な最適化（実行速度優先）-Ot オプションを指定しているときに有効です。

(3) ファイル・マージ・オプション

-Om

複数ファイル同時指定時に，ファイルのマージを行います。コンパイル時間は遅くなりますが，最適化オプション -O，-Os，および -Ot と同時に指定することで，関数間最適化などを最適化適用範囲を広くすることができます。ただし，ソース・デバッグは難しくなります。

(4) インライン展開最適化制御オプション

-Wp,-G*num*

【PM+】

インライン展開対象関数のスタック・サイズを，中間言語での大きさ *num* に制限し，*num* より大きいものはインライン展開しません。*num* の目安については，後述の -Wp,-I[=file] オプションを参照してください。このオプションを指定しない場合，-Wp,-G32 が指定されたものとみなします。

-Wp,-N*num*

インライン展開対象関数の中間言語サイズを *num* に制限し，*num* より大きいものはインライン展開しません。*num* の目安については，後述の -Wp,-I[=file] オプションを参照してください。このオプションを指定しない場合，より高度な最適化（実行速度優先）オプション指定時には -Wp,-N128 が指定されたものとみなし，それ以外では -Wp,-N24 が指定されたものとみなします。

-Wp,-S

【PM+】

一度しか参照されない静的な関数を無条件にインライン展開します。

-Wp,-l[=file]

【PM+】

関数の情報を標準出力に出力，または file に追加出力します。表示される情報は，上記の -Wp,-G，-Wp,-N オプションで指定する値の目安となります。たとえば，スタック・サイズでは，呼び出された関数の値が -Wp,-G で指定した値以下であればインライン展開されます。また，コード・サイズでは，呼び出された関数の値が -Wp,-N で指定した値以下であればインライン展開されます。

なお，このオプションによって出力されるスタック・サイズは，あくまでもプリオプティマイザが出力する中間言語でのサイズであるため，関数が実際に使用するスタック・サイズとは異なります。

-Wp,-inline

【PM+】

#pragma inline 指定した関数のみインライン展開します。-Ot 指定時にはコンパイラが自動判別しインライン展開を行います。ユーザが指定した関数のみ展開する場合は，このオプションを指定してください。

-Wp,-no_inline

【PM+】

#pragma inline 指定した関数を含む，すべての関数のインライン展開を抑止します。-Ot 指定時に，インライン展開機能をすべて抑止する場合に有効です。

-Wp,-r[*funcname*]

関数 *funcname* をエントリ関数として，そこから呼び出された関数のうち，展開後の不要な関数を削除します。*funcname* は，C 言語で記述された関数の先頭に ‘ _ ’ を付けて指定します。*funcname* を指定しない場合は，“ _main ” が指定されたものとみなします。

なお，アセンブリ言語ソースによってのみ呼び出される関数は，呼び出されていることが認識できないため，不要な関数として削除されます。

ただし，割り込み関数とリアルタイム OS 用のタスクは，関数削除の対象から除外されています。

インライン展開とそのオプションの関係については “ CA850 ユーザーズ・マニュアル C 言語編 ” も参照してください。

(5) ループ展開最適化制御オプション

-Wo,-Ol[*num*]

【PM+】

for，while などで，ループを *num* 回展開します。実行速度優先最適化の場合にのみ指定できます。実行回数が N 回（N は定数）のループの実行と *num* 回展開されたコードを含むループの実行に変換されます。

ただし，展開後のコード・サイズが大きいかループの実行回数が少ない場合は，展開数が少なくなったり展開されない場合があります。また，内側にループを含むような複雑な構造のループは，展開されない場合があります。*num* に 0，または 1 を指定した場合，展開が抑止されます[注]。また，*num* を指定しない場合，4 が指定されたものとみなします。なお，*num* は 10 進数で指定してください。

注　より高度な最適化（実行速度優先）指定時で，ループ展開を行いたくない場合に有効です。

例

実行回数が 10 のループを 4 回展開する場合

```
i = 0;
while(i < 10) {
        /* 処理 */
        ++i;
}
```

```
i = 0;
/* 処理 */
i =1;
/* 処理 */
i = 2;
while(i < 10) {
        /* 処理 */
        ++i;
        /* 処理 */
        ++i;
        /* 処理 */
        ++i;
        /* 処理 */
        ++i;
}
```

-Wo,-Xlo

【PM+】

ループ展開数を -Wo,-Ol*num* で指定した回数で固定してループ展開します。

(6) strcpy，strcmp 展開オプション

-Xi

【PM+】

配列（文字列を含む），および構造体の整列条件を 4 バイトとし，関数 strcpy()，または strcmp() の呼び出しをインライン展開します。オブジェクトの実行速度は高速になりますが，コード・サイズは増大します。このオプションは，strcpy()，または strcmp() の第二引数が文字列の場合にのみ変換します。また，第一引数は，プログラム側で 4 バイトに整列されている必要があります（第二引数は文字列のため，ca850 が整列しています）。このオプションは，-Xpack オプションと同時指定はできません。

(7) 外部変数ソート・オプション

-Wo,-Op[=*file*]

【PM+】

const ／ sconst セクション以外のセクションに配置されている外部変数をアライメントの大きい順に並び替えます。中間言語ファイル *file* を指定した場合は，ソース・ファイル中の外部リンケージを持つ const ／ sconst セクション以外のセクションに配置されている変数の定義および仮定義を *file* に移動します。移動後のソース・ファイルの変数の定義および仮定義は，宣言と同じ扱いになります。最初に *file* が存在しなくてもエラーにはなりません。

(8) 分岐命令制御オプション

-Wo,-XFo

【PM+】

分岐命令をコード・サイズ優先で並べてコードを出力します。

ただし，ソース・デバッグが難しくなります。このオプションは -Og ／ -O ／ -Os ／ -Ot 指定時に有効になります。このオプションを省略した場合は，分岐命令に対してデバッグ情報を優先したコードを出力します。

3.5.6 生成コード制御

次に生成コード制御オプションを示します。

(1) レジスタ使用制御オプション

-*rnum*=*sym*

【PM+】

指定した外部変数 *sym* をレジスタ r*num* に割り付けます。*num* は -reg オプションを指定して空けたマスク・レジスタ以外のレジスタを指定します。*sym* は外部変数名（先頭の“ _ ”を除く）で，volatile 変数，アドレス演算子を使用した変数，集成体，配列，内部リンケージを持つ変数，および周辺 I/O レジスタは指定できません。

-reg*n*

【PM+】

ca850 が使用するレジスタを *n* 本に制限します（レジスタ・モードを *n* とします）。

n の値として指定できるのは次のとおりです。

表 3 － 1 レジスタ・モード

レジスタ・モード（*n*）	作業用レジスタ	レジスタ変数用レジスタ
22	r10 ～ r14	r25 ～ r29
26	r10 ～ r16	r23 ～ r29
32	r10 ～ r19	r20 ～ r29

なお，このオプションは，ソース・ファイルごとの指定では選択できません。常に全体のオプションとして設定します。また，このオプションによる設定はリンカにも認識されるため，適切なモードのライブラリが参照されます。

このオプションを省略した場合，32 本を指定（32 レジスタ・モードを指定）したものとみなします。

このオプションを指定することで，ソフトウエア・レジスタ・バンク機能のレジスタ・モードを切り替えることができます。

-Xmask_reg

【PM+】

マスク・レジスタ機能を使用します。この機能を使用すると，ca850 は r20 に 8 ビットのマスク値 0xff，r21 に 16 ビットのマスク値 0xffff が設定されているものとしてコードを出力します。マスク・レジスタ（r20，r21）へのマスク値の設定は，スタート・アップ・ルーチン等，ユーザ・プログラムで行う必要があります。

V850 マイクロコントローラでは，バイト・データ，ハーフワード・データをメモリからレジスタにロードする場合，最上位ビットの値によりワード長へ符号拡張します。このため，unsigned char，unsigned short 型データの演算では，上位ビットのマスク・コードが生成される場合があります。また，演算結果をレジスタ変数へストアする場合，符号なしバイト・データ，符号なしハーフワード・データでは，上位ビットをクリアするためにマスク・コードが生成されます。どちらの場合も，ワード・データに切り替えれば回避できますが，ワード・データにできず，マスク・コードが生成される場合，マスク・レジスタ機能を用いることにより，コード・サイズの削減ができます。

ただし，マスク・レジスタ機能を利用するかどうかの判断には，利用する側で次の点を十分考慮する必要があります。

- マスク・コードが多く出力されるプログラムであるか。
- マスク・レジスタとして使用するため，レジスタ変数用レジスタが 2 本少なくなるが，その影響はないか。

なお，このオプションを指定したとき，マスク・レジスタを使用したオブジェクトとしないオブジェクトが混在する場合，ld850 でエラーとなります。また，32 レジスタ・モードのとき，ld850 に -mask_reg が渡されます。これにより，リンカによる標準ライブラリの検索は，標準フォルダより先に，マスク・レジスタ用フォルダ（lib850\r32mak）で行います。

マスク・レジスタについての詳細は “ CA850 ユーザーズ・マニュアル C 言語編 ” を参照してください。

(2) プロローグ／エピローグ処理制御オプション

-Xpro_epi_runtime[=on|=off]

【PM+】

関数のプロローグ／エピローグ処理をランタイム・ライブラリ呼び出しによる処理にするかどうかを設定します。on にした場合，関数のプロローグ／エピローグ処理がランタイム・ライブラリ呼び出しになります。[=on][=off] の指定をしなかった場合，[=on] が指定されたものと見なします。デフォルト時の動作は on となり，このオプションで [=off] を指定した場合，または -Ot オプションを指定した場合のみ off となります。

(3) 変数配置制御オプション

-G*num*

【PM+】

num バイト以下のデータを .sdata セクション，または .sbss セクションに配置します。このオプションを省略した場合，すべてのデータが，.sdata ／ .sbss セクションに配置されます。

ただし，#pragma section 指令，またはセクション・ファイルで .sdata ／ .sbss セクションが指定されたデータは，そのサイズに関係なく .sdata ／ .sbss セクションに配置します。

なお，*num* は 10 進数で指定してください。*num* に設定する値の目安は，ld850 の -A オプションで出力されます。

.sdata セクション，.sbss セクションについての詳細は“ CA850 ユーザーズ・マニュアル C 言語編 ”を参照してください。

-Xsconst[=*num*]

【PM+】

const 属性のデータ , 文字列リテラルを .sconst セクションに配置します。*num* を指定した場合，*num* バイト以下のデータを .sconst セクション配置し，*num* を省略した場合，そのサイズに関係なく配置します。

なお，*num* は 10 進数で指定してください。ファイルごとに異なるオプションを指定すると，変数の配置，および参照方法が異なるコードを生成しリンク時にもエラー，または警告メッセージが出力されることがあります（PM+ で指定する場合は，ファイルごとに指定はできません）。

-Xcre_sec_data[=*outfile*]

【PM+】

セクション・ファイル・ジェネレータで使用する変数の頻度情報ファイルを出力します。

(a) *outfile* にファイル名を指定した場合

指定ファイル名でカレント・フォルダに *outfile* を保存します。

(b) *outfile* にフォルダを指定した場合

指定フォルダに .c を .sec で置き換えたファイル名で保存します。

(c) =*outfile* を省略した場合

カレント・フォルダに .c を .sec で置き換えたファイル名で保存します。

(d) 出力ファイルが複数の場合

outfile に指定したフォルダを作成し，.c を .sec で置き換えたそれぞれのファイル名で保存します。

なお，C 言語ソース・ファイルが複数存在し，それぞれのファイルに対してファイル名を指定して頻度情報ファイルを生成したい場合，PM+ では，ソース・ファイル一覧から C 言語ソース・ファイルに対してオプション設定用ダイアログを表示し，頻度情報ファイルの生成でファイル名を指定します。コマンド・ラインでは，各 C 言語ソース・ファイルに対して，“ =*outfile* ”付きでこのオプションを指定します。その際，C 言語ソース・ファイルは，(-c 指定で）1 つずつ指定します。

変数の頻度情報ファイルは，C 言語ソース・ファイル内の変数に対する ld，st 命令でのアクセス頻度情報を出力します。アセンブリ言語ソース・ファイルに対しては何も行いません。

なお，このオプションと -Xcre_sec_data_only オプションを同時に指定した場合は，-Xcre_sec_data_only オプションが優先されます。

-Xcre_sec_data_only[=*outfile*]

セクション・ファイル・ジェネレータで使用する，変数の頻度情報ファイルを出力します。

ただし，-Xcre_sec_data **と違い，変数の頻度情報ファイルのみを出力するため，オブジェクトの生成までは行われません。頻度情報ファイルのみを出力したい場合に使用します。**

(a) *outfile* **にファイル名を指定した場合**

指定ファイル名でカレント・フォルダに *outfile* **を保存します。**

(b) *outfile* **にフォルダを指定した場合**

指定フォルダに .c **を** .sec **で置き換えたファイル名で保存します。**

(c) =*outfile* **を省略した場合**

カレント・フォルダに .c **を** .sec **で置き換えたファイル名で保存します。**

(d) **出力ファイルが複数の場合**

outfile **に指定したフォルダを作成し，**.c **を** .sec **で置き換えたそれぞれのファイル名で保存します。**

なお，C **言語ソース・ファイルが複数存在し，それぞれのファイルに対してファイル名を指定して頻度情報ファイルを生成したい場合，**PM+ **では，ソース・ファイル一覧から** C **言語ソース・ファイルに対してオプション設定用ダイアログを表示し，頻度情報ファイルの生成でファイル名を指定します。コマンド・ラインでは，各** C **言語ソース・ファイルに対して，“** =*outfile* **” 付きでこのオプションを指定します。その際，**C **言語ソース・ファイルは，(**-c **指定で) 1 つずつ指定します。変数の頻度情報ファイルは，**C **言語ソース・ファイル内の変数に対する** ld**，**st **命令でのアクセス頻度情報を出力します。アセンブリ言語ソース・ファイルに対しては何も行いません。**

-Xsec_file=*file*

【PM+**】**

ca850 **起動時にデータのセクション割り当てを指定するためのセクション・ファイル（「9.1 セクション・ファイル」参照）名を指定します。必ずファイル名を指定してください。このオプションを複数指定し，複数のセクション・ファイルを入力することもできます。**

(4) 型制御オプション

-Xbitfield=*string*

【PM+】

型指定子（signed，unsigned）の付かない単なる int 型のビット・フィールドに対し，符号付きとするか符号なしとするかを指定します。

string に指定できるものは，次のとおりです。

s	符号付きとして扱う
signed	符号付きとして扱う
u	符号なしとして扱う
unsigned	符号なしとして扱う

なお，符号なしとして扱う場合，警告メッセージを出力します。このオプションを省略した場合，符号付きとして扱います。

-Xchar=*string*

【PM+】

型指定子（signed，unsigned）の付かない単なる char 型に対し，符号付きとするか符号なしとするかを指定します。

string に指定できるものは，次のとおりです。

s	符号付きとして扱う
signed	符号付きとして扱う
u	符号なしとして扱う
unsigned	符号なしとして扱う

このオプションを省略した場合，符号付きとして扱います。

-Xenum_type=*string*

【PM+】

列挙型に対し，どの整数型と整合するかを指定します。

string に指定できるものは，次のとおりです。

char	char として扱う
uchar	unsigned char 符号付きとして扱う
short	short として扱う
ushort	unsigned short として扱う

このオプションを省略した場合，signed int として扱います。

(5) switch-case 文出力コード制御オプション

-Xcase=*string*

【PM+】

switch 文のコード出力方式を指定します。*string* に指定できるものは，次のとおりです。

ifelse	case 文の並びに沿った if-else 文と同じ形で出力します。 頻度が多い順に case 文を書いているときやラベル数が少ないときは，これを選択します。上から順に比較するので，頻繁に合致する case 文を先に記述すると余計な比較が減り，実行速度向上につながります。
binary	バイナリ・サーチ形式で出力します。 バイナリ・サーチ・アルゴリズムに用いて合致する case 文を探します。ラベル数が多いときにこれを選択すると，どの case 文も同じくらいの速さで見つけることができます。
table	テーブル・ジャンプ方式で出力します。 case 文の値を基にインデックス化したテーブルを参照し，switch 文の値により case ラベルを選択し，処理を行います。どの case 文にも同じくらい速く分岐します。ただし，case 値が連続していないときは無駄な領域ができます。

このオプションを省略した場合，コンパイラが最適と思われる形式を自動判断します。

-Xword_switch

【PM+】

switch 文の case ラベルに対する分岐テーブルを，1 ラベルあたり 4 バイトで生成します。長い switch 文のためコンパイル・エラーとなる場合，このオプションを指定します。このオプションを省略した場合，分岐テーブルを 2 バイトで生成します。

(6) 構造体パッキング制御オプション

-Xbyte

【PM+】

構造体の間接アドレス・アクセスをバイト単位でアクセスします。構造体パッキング機能で，制限にかかるような場合に使用します。

-Xpack=*num*

【PM+】

このオプションを用いることにより，構造体メンバをメンバの型に応じてアライメントすることなく，指定したアライメントを用いることができます。データ・サイズは小さくすることができますが，コード・サイズは大きくなります。*num* には，1，2，4，8 が指定できます。デフォルトは 8 です。[注]

C 言語ソース・ファイル中に #pragma 指令で構造体パッキング指定がある場合に，このオプションを指定した場合，最初の #pragma 指令が出現するまでは，オプション指定値がすべての構造体に適用されます。それ以降は，#pragma 指令の値が適用されます。

ただし，#pragma 指令の出現後でも指定がデフォルトになった部分は，オプション指定値が適用されます。このオプションは，-Xi オプションと同時指定はできません。

また，このオプションには，次の注意点があります。

(a) C 言語ソース・ファイル中に #pragma 指令で構造体パッキング指定がある時に -Xpack オプションを指定した場合，最初の #pragma pack 指令が出現するまではオプション指定値がすべての構造体に適用されます。それ以降は #pragma 指令の値が適用されます。
ただし，#pragma 指令の出現後でも指定がデフォルトになった部分は，オプション指定値が適用されます。

また，V850 ／ V850Ex ／ V850E2 コア製品ミス・アライン・アクセス禁止の設定の CPU をご使用の場合，次の制限があります。この制限は，#pragma pack についても同様です。

(a) 構造体メンバのアドレスを取得すると正しく取得できません。
(b) ビットフィールドへのアクセスは，そのメンバの型で読み込むため，データ領域もアクセスします。ビットフィールドの幅が，メンバの型以下の場合，メンバの型で読み込むのでオブジェクトの外部にアクセスします。実行上通常は問題がありませんが，I/O などがマップされていた場合，不正なアクセスとなる場合があります。

構造体パッキングについての詳細は “ CA850 ユーザーズ・マニュアル C 言語編 ” を参照してください。

注　本バージョンでは *num* の値が “ 4 ” と “ 8 ” のときの動作は同じになります。

(7) far jump 出力制御オプション

`-Xfar_jump=`*file*

`-Xfar_jump` *file*

【PM+】

file に記述された関数への分岐に対して，jmp 命令を使用します。関数本体が，jarl，および jr 命令では分岐できない範囲（± 1M バイト以上）にあり，ld850 がエラーを出力する場合，このオプションを用いてコンパイルし直します。なお，ファイル名には拡張子が必要となります。推奨する拡張子は “ .fjp ” です。

Flash ／外付け ROM 再リンク機能でブート側からフラッシュ側の関数を呼び出す場合には，このオプションは指定できません。詳細は「5.6 フラッシュ／外付け ROM 再リンク機能」を参照してください。

`-Xj`

【PM+】

C 言語で定義された通常の割り込み関数に対し，jmp 命令を用います。関数本体が，jr 命令では分岐できない範囲（± 1M バイト以上）にあり，ld850 がエラーを出力する場合，このオプションを用いてコンパイルし直します。このオプションを省略した場合，jr 命令を用います。

Flash ／外付け ROM 再リンク機能でブート側からフラッシュ側の関数を呼び出す場合には，このオプションは指定できません。詳細は「5.6 フラッシュ／外付け ROM 再リンク機能」を参照してください。

(8) コメント出力オプション

`-Xc`

【PM+】

出力するアセンブリ言語ソース・ファイル中に C 言語ソース・プログラムをコメントとして出力します。ただし，出力されるコメントは，あくまで参考であり，厳密にはコードと対応していない場合もあります。たとえば，グローバル変数とローカル変数，関数宣言などのコメントの出力位置がずれることがあります。また，最適化によりコードが削除され，コメントのみが残ることもあります。

本オプションは，-S，-a，-Fs，または -Fv のいずれかが指定されている必要があります。

(9) ANSI 規約オプション

-Xe

【PM+】

16 ビット・データ以下の整数に対し mulh，divh 命令を利用せず，ランタイム・ライブラリ ___mul ／ ___mulu，___div ／ ___divu を利用します。実行速度は遅くなりますが，ANSI 規格に厳密な乗除算処理を行います。このオプションを省略した場合，mulh，divh 命令を利用します。

CA850 におけるランタイム・ライブラリは，V850 マイクロコントローラのアーキテクチャにはない命令について，ANSI 規格を満たすために CA850 の標準ライブラリで用意されているものです。

ランタイム・ライブラリについての詳細は“ CA850 ユーザーズ・マニュアル C 言語編 ”を参照してください。

-Xdefvar

【PM+】

変数の仮定義を定義として扱います。

-ansi

【PM+】

ca850 の処理を厳密に ANSI 規格にあわせ，規格に反する記述に対してエラーや警告メッセージを出力します。_asm 形式以外の拡張記述は認められます。また，このオプション指定時は，マクロ名 __STDC__ を定義します。このオプションを省略した場合，従来の C 言語の仕様との両立性を持たせ，警告メッセージを出力して処理を続行します。

言語仕様に厳密なコンパイル時の処理は次のようになります。

(1) トライグラフ系列

トライグラフを置換する。このオプションを指定しなかった場合，置換しない。

(2) ビット・フィールド

ビット・フィールドに int 型以外の型を指定した場合，エラーとする。このオプションを指定しなかった場合，警告メッセージを出力して許可する。

(3) 引数のスコープ

関数引数と同名の自動変数を宣言した場合，二重定義エラーとする。このオプションを指定しなかった場合，警告メッセージを出力して自動変数を有効とする。

(4) ポインタの代入

(a) 汎整数型変数へのポインタ型数値の代入はエラーとする。このオプションを指定しなかった場合，警告メッセージを出力し，キャストして代入する。

(b) 異なる型を指すポインタ同士の代入はエラーとする。このオプションを指定しなかった場合，警告メッセージを出力して許可する。

(5) 型変換

左辺値でない配列のポインタへの変換はエラーとする。このオプションを指定しなかった場合，警告メッセージを出力して許可する。

(6) 比較演算子

算術型変数とポインタの比較はエラーとする。このオプションを指定しなかった場合，警告メッセージを出力して許可する。

(7) **条件演算子**

第 2 式と第 3 式がともに汎整数型，同じ構造体，同じ共用体，または代入先と同じ型へのポインタ型のいずれでもない場合，エラーとする。このオプションを指定しなかった場合，警告メッセージを出力し，キャストして代入する。

(8) **# 行番号**

エラーとする。このオプションを指定しなかった場合，“ #line **行番号 ” と同様に扱う。**

(9) **行の途中の** '#' **文字**

エラーとする。このオプションを指定しなかった場合，警告メッセージを出力して許可する。

(10) _asm

警告メッセージを出力して関数呼び出しとして扱う。ただし __asm（'_'2 **つ）は有効。このオプションを指定しなかった場合，アセンブラ挿入として扱う。**

(11) __STDC__

値が 1 **のマクロとして定義する。このオプションを指定しなかった場合，マクロとして定義しない。**

(12) 2 **進定数**

使用不可。このオプションを指定しなかった場合，“ 0b **”，または “** 0B **” と，その後ろに続く** 1 **個以上の “** 0 **”，または “** 1 **” の数字の並びを** 2 **進定数とします。**

3.5.7 日本語文字列制御

次に日本語文字列制御オプションを示します。

-Xk=*code*

【PM+】

入力ファイル中の日本語のコメント，文字列に対し，使用する文字コードを指定します。指定したコード以外のコードが存在する場合，エラー・メッセージを出力します。

code に指定できるものは，次のとおりです。

e	EUC
euc	EUC
n	コードを保証しない
none	コードを保証しない
s	シフト JIS
sjis	シフト JIS

このオプションを省略した場合，シフト JIS とします。

-Xkt=*code*

【PM+】

日本語文字列を，指定したコードに変換して出力します。アプリケーション開発時に使用した漢字コードを，ターゲットで変更したい場合に有効です。

code に指定できるものは，次のとおりです。

e	EUC
euc	EUC
n	コードを保証しない
none	コードを保証しない
s	シフト JIS
sjis	シフト JIS

なお，-Xc 指定により出力されるコメントは，このオプションの指定に関わらず，変換しません。このオプションを省略した場合，コードを変換しません。また，このオプションによりコードを変換したい入力ファイル中で，デフォルト（シフト JIS）以外のコードを使用している場合，[C 言語と漢字] タブの “ ソースの漢字コード ” により，使用しているコードを指定してください。

3.5.8 ライブラリ指定

次にライブラリ指定オプションを示します。

-L*dir*

ライブラリをフォルダ *dir*，標準フォルダの順で検索します。このオプションを省略した場合，標準フォルダのみ検索します。標準フォルダは，“ インストール・フォルダ \lib850 フォルダ ”，および “ インストール・フォルダ \lib850\r32 フォルダ ” です。ただし，レジスタ・モードが指定された場合，r32 フォルダの代わりに，同じ並びの r22，または r26 フォルダで検索します。

ld850 の -L オプションを参照してください。

-R *file*

ld850 まで起動する場合，使用するスタート・アップ・モジュールを *file* とするよう ld850 に指示します。このオプションを省略した場合，標準フォルダにある crtN.o，または crtE.o をスタート・アップ・モジュールとします。標準フォルダは，“ インストール・フォルダ \lib850\r32（r26，r22）” です。

-l*string*

ld850 で参照する，アーカイブ・ファイルを指定します。このオプションを省略した場合，参照しません。ただし，ca850 から ld850 を起動する場合，ca850 は自動的に標準ライブラリ，および数学ライブラリのリンク（-lm -lc）指定を ld850 に渡します。

アーカイブ・ファイルの指定方法については，ld850 のライブラリ指定（-l）オプションを参照してください。

3.5.9 警告メッセージ制御

次に警告メッセージ制御オプションを示します。

-w*num*

【PM+】

警告メッセージのレベルを指定します。

num に指定できる数字は，次のとおりです。

0	メッセージを抑止する
1	通常の警告メッセージを出力する
2	詳細なメッセージを出力する

num を省略した場合，-w0 が指定されたものとみなします。このオプションを省略した場合，-w1 が指定されたものとみなします。

-w*string*+

-w*string*-

レベルに関わりなく，項目ごとに警告メッセージの出力，および抑止を指定します。‘ + ’を付けた場合メッセージを出力し，‘ - ’を付けた場合メッセージを抑止します。

string に指定できる文字列は，次のとおりです。

bitfield_align	ビット・フィールドのメンバが整列条件の境界を越えるため，次の境界から割り当てられた場合
bitfield_type	ビット・フィールドに対し，ANSI 仕様で指定することのできない型が指定された場合
callnodecl	宣言のない関数が呼ばれた場合
nopic	自動変数でない変数のアドレス，または関数のアドレスを用いたポインタ型外部変数の初期化の場合
pragma	実行不可能な #pragma 記述が現れた場合
sharp	ソース行中に＃文字が現れた場合

なお，‘ + ’，および ‘ - ’ を付けない場合，エラーとなります。このオプションを省略した場合，-w*num* のレベル指定に従います。

-won=*num*[,*num*]...

-won=*num1*-*num2*[,*num3*-*num4*]...

【PM+】

num で指定した番号の警告メッセージを出力するようにします。*num* には 2000 番台の警告メッセージが指定できます。W2042 の警告メッセージを出力する場合は，-won=2042 になります。*num1*-*num2* の形式で指定すると，*num1* から *num2* までの警告メッセージを指定したことになります。*num* は省略できません。また，ca850 にない警告メッセージ番号を指定すると，警告メッセージが出力されます。

-woff=*num*[,*num*]...

-woff=*num1*-*num2*[,*num3*-*num4*]...

【PM+】

num **で指定した番号の警告メッセージを抑制します。***num* **には** 2000 **番台の警告メッセージが指定できます。**W2042 **の警告メッセージを抑制する場合は，**-woff=2042 **になります。***num1*-*num2* **の形式で指定すると，***num1* **から** *num2* **までの警告メッセージを指定したことになります。***num* **は省略できません。また，**ca850 **にない警告メッセージ番号を指定すると，警告メッセージが出力されます。**

3.5.10 その他

次にその他オプションを示します。

(1) コマンド・ファイル指定オプション

`@cfile`

cfile をコマンド・ファイル（「3.7.2 コマンド・ファイル」参照）として扱います。これにより，オプション文字列の長さの制限を意識せずに済みます。コマンド・ファイルでは，指定する引数を複数行に分けて記述できますが，オプションやファイル名などが 2 行に分かれないようにしてください。

このオプションを省略した場合，コマンド・ファイルがないものとみなします。

(2) CPU 不具合パッチ・オプション

`-Xv850patch[=num]`

CPU の障害に対応したコードを出力するため，ca850 が出力するアセンブリ言語ソース・ファイルに対し，*num* に応じて，as850 に -p[*num*] オプション指定を指示します（「4.7.2 CPU 不具合の回避オプション」参照）。*num* には 1，2，3，4，4a，5，6，7，8，9，10，11 が指定できます。5 ～ 10 は V850Ex コアでのみ有効です。“ =*num* ” を省略した場合，*num* に 1，2，3，5，6，7，8，9，10 が指定されたものとみなします。

このオプションは，CPU の障害回避のためのオプションです。使用している CPU に該当する障害であるかどうかは，CPU 添付の資料を参照してください。

なお，-Xv850patch=11 オプションのみ ca850 で処理します。-Xv850patch=11 オプションを指定すると，次の命令を出力しなくなります。

- set1 ／ clr1 ／ not1
- V850Ex ／ V850E2 コアのミスアライン・アクセス（構造体パッキング時）

ただし，asm 文とアセンブリ言語ソース・ファイルに関してはチェックを行わないので，これらの箇所で使用した場合はそのまま出力されます。

また，-Xv850patch=11 オプションを指定して，プログラム中で周辺 I/O レジスタへのビット・アクセスの記述をした場合には，周辺 I/O レジスタへワード（4byte）単位のアクセスになります。ビット・アクセスではなく，バイト／ハーフワード単位での操作の記述に変更してください。

CPU コアと，バッチ・オプションに対応する不具合は，次のとおりです（保守品と廃品種を除く，最新バージョンの μPD(F)703xxx の場合）。

なお，使用している CPU に該当する障害であるかどうかは，CPU の資料を参照してください。

表 3 － 2　CPU コアと -Xv850patch オプションに対応する不具合

CPU コア	-Xv850patch=11
V850 コア	－
V850E/MS1	×
V850E1 コア	×
V850ES コア	×

備考　×：該当

○：修正済み（保守品と廃品種を除く，最新バージョンの μPD(F)703xxx の場合）

－：非該当

3.5.11 各モジュール

ca850 から各モジュールにオプションを渡すことができます。

-Wx,*option*

option をモジュール *x* に対するオプションとしてを渡します。*option* がカンマを含む場合は，そのカンマで区切られた複数のオプションとして与えられます。

モジュール *x* に指定できるものは，次のとおりです。

p	プリオプティマイザ（popt）
o	広域最適化部（opt）
i	機種依存最適化部（impr）
a	アセンブラ（as850）
l	リンカ（ld850）

(1) プリオプティマイザ（popt）

-Wp,-D

コンパイル時のメモリ使用量を減らすことができます。

-Wp,-G*num*

インライン展開対象関数のスタック・サイズを，中間言語での大きさ *num* に制限し，*num* より大きいものはインライン展開しません。*num* の目安については，後述の -Wp,-I[=file] オプションを参照してください。このオプションを指定しない場合，-Wp,-G32 が指定されたものとみなします。

-Wp,-N*num*

インライン展開対象関数の中間言語サイズを *num* に制限し，*num* より大きいものはインライン展開しません。*num* の目安については，後述の -Wp,-I[=file] オプションを参照してください。このオプションを指定しない場合，より高度な最適化（実行速度優先）オプション指定時には -Wp,-N128 が指定されたものとみなし，それ以外では -Wp,-N24 が指定されたものとみなします。

-Wp,-S

一度しか参照されない静的な関数を無条件にインライン展開します。

-Wp,-l[=file]

関数の情報を標準出力に出力，または file に追加出力します。表示される情報は，上記の -Wp,-G，-Wp,-N オプションで指定する値の目安となります。たとえば，スタック・サイズでは，呼び出された関数の値が -Wp,-G で指定した値以下であればインライン展開されます。また，コード・サイズでは，呼び出された関数の値が -Wp,-N で指定した値以下であればインライン展開されます。

なお，このオプションによって出力されるスタック・サイズは，あくまでもプリオプティマイザが出力する中間言語でのサイズであるため，関数が実際に使用するスタック・サイズとは異なります。

-Wp,-r[_*funcname*]

funcname をエントリ関数として，そこから呼び出された関数のうち展開後に不要な関数を削除します。*funcname* は，関数名の先頭に ‘ _ ’ を付けて指定します。*funcname* を指定しない場合，“ _main ” が指定されたものとみなします。

なお，アセンブラ文によってのみ呼び出されている関数は，呼び出されていることが認識されず，不要な関数として削除されます。

ただし，割り込み関数とリアルタイム OS 用のタスクは，関数削除の対象から除外されています。

-Wp,-inline

#pragma inline 指定された関数のみインライン展開します。

-Wp,-no_inline

#pragma inline 指定された関数を含むすべてのインライン展開を抑止します。

(2) 広域最適化部（opt）

-Wo,-Ol[*num*]

for，while などで，ループを *num* 回展開します。実行速度優先最適化の場合にのみ指定できます。実行回数が N 回（N は定数）のループの実行と *num* 回展開されたコードを含むループの実行に変換されます。

ただし，展開後のコード・サイズが大きいか，ループの実行回数が少ない場合は，展開数が少なくなったり展開されない場合があります。また，内側にループを含むような複雑な構造のループは，展開されない場合があります。*num* に 0，または 1 を指定した場合，展開が抑止されます[注]。また，*num* を指定しない場合，4 が指定されたものとみなします。なお，*num* は 10 進数で指定してください。

注　より高度な最適化（実行速度優先）指定時で，ループ展開を行いたくない場合に有効です。

例

実行回数が 10 のループを 4 回展開する場合

```
i = 0;
while(i < 10) {
        /* 処理 */
        ++i;
}
```

```
i = 0;
/* 処理 */
i =1;
/* 処理 */
i = 2;
while(i < 10) {
        /* 処理 */
        ++i;
        /* 処理 */
        ++i;
        /* 処理 */
        ++i;
        /* 処理 */
        ++i;
}
```

-Wo,-Op[=*file*]

【PM+】

const ／ sconst セクション以外のセクションに配置されている外部変数をアライメントの大きい順に並び替えます。中間言語ファイル *file* を指定した場合は，ソース・ファイル中の外部リンケージを持つ const ／ sconst セクション以外のセクションに配置されている変数の定義および仮定義を *file* に移動します。移動後のソース・ファイルの変数の定義および仮定義は，宣言と同じ扱いになります。最初に *file* が存在しなくてもエラーにはなりません。

-Wo,-XFo

【PM+】

分岐に関して，コード・サイズを優先したコードを出力します。

ただし，デバッグ情報への影響があります。このオプションは，-Og ／ -O ／ -Os ／ -Ot 指定時に有効になります。

このオプションを省略した場合，分岐に関して，デバッグ情報を優先したコードを出力します。

-Wo,-Xlo

ループ展開を CA850 Ver.2.02 以前のバージョンの条件で展開します。

(3) 機種依存最適化部（impr）

`-Wi,-D`

コンパイル時のメモリ使用量を減らすことができます。

ただし，コンパイル速度は低化します。メモリを非常に多く使い，コンパイルが正常にできなくなるような場合に指定します。

`-Wi,-O4`

【PM+】

データ・フロー解析を緻密に行い，次の最適化を実行します。

(a) 分岐命令をまたいだレジスタの最適化

(b) 絶対値演算の最適化

(c) 分岐命令をまたいだ cmp 命令の最適化

(d) 分岐命令をまたいだ復帰命令の最適化

ただし，コンパイル速度はかなり遅くなります。最適化オプション -O，-Os，または -Ot 指定をした上で，さらにデータ・フロー解析を強力に行いたい場合のみ指定してください。

`-Wi,-P`

【PM+】

ラベルを整列する最適化を抑制します。これにより，コード・サイズを小さくできます。

(4) アセンブラ（as850）

「4.4 オプションの種類と機能」を参照。

(5) リンカ（ld850）

「5.3 オプションの種類と機能」を参照。

3.6 PM+ での設定

この節では，コンパイラのオプションを設定する各ダイアログについて説明します。

[コンパイラ共通オプションの設定] ダイアログは，コンパイラで共通するオプションを設定するダイアログです。次の操作で表示します。

(1) [ツール] メニュー→ [コンパイラ共通オプションの設定] を選択

[コンパイラオプションの設定] ダイアログは，対象プロジェクトの C 言語ソース・ファイルに対して，ca850 のコマンド・オプションを設定するダイアログです。[プロジェクト] メニューからプロジェクトを設定後，次のいずれかの操作で表示します。

- 対象プロジェクト全体の C 言語ソース・ファイルに設定する場合

(1) [ツール] メニュー→ [コンパイラオプションの設定] を選択

- 個別の C 言語ソース・ファイルに設定する場合

(1) PM+ のプロジェクト・ウインドウ内で，オプションを設定したいソース・ファイル名を選択
(2) 右クリック・メニュー→ [個別コンパイラオプションの設定] を選択

個別の設定が行われている場合，そのファイルに対しては，全体のオプション設定が無効になります。個別オプションが設定されたファイルには，プロジェクト・ウインドウに表示される “ ソース・ファイル ” のファイル名の先頭にあるアイコンが緑色に変化します。
個別のオプション設定を無効にし，全体のオプション設定を有効にしたい場合は，そのファイルを選択して右クリックで表示される [個別コンパイラオプションの解除], または [コンパイラオプションの設定] ダイアログの [一般] タブ，[その他] タブ，および [差分] タブに追加されている [ソースオプションの削除] ボタンをクリックします。これにより個別の設定が削除されます。

なお，ダイアログ上の “ [] ” 内に表示されたオプション名は，コマンド・プロンプトから起動する場合のオプション名です。
また，出力オブジェクト名を指定したい場合など，コマンド・プロンプトからの起動時にのみ指定可能なオプションもあります。
ビルド・モードの切り替えは，[コンパイラオプションの設定] ダイアログの [最適化とデバッグ情報] タブの “ デバッグ情報の生成 [-g] ” を選択するとデバッグ情報を出力するモードになります。

インクルード・ファイルのパスの設定，および変更を行った場合には，PM+ で [ビルド] メニュー→ [依存関係の更新] を選択後，[プロジェクト] メニュー→ [メイクファイルのエクスポート] の選択を行ってください。

3.6.1 [コンパイラ共通オプションの設定] ダイアログ

[コンパイラ共通オプションの設定] ダイアログの上部には，次の 6 個のタブが表示されており，タブの選択により，ダイアログの表示内容が変わります。

表 3 － 3 [コンパイラ共通オプションの設定] ダイアログ

タブ	内容
[ファイル]	出力ファイルに関する設定
[スタートアップ]	スタート・アップ・ファイルに関する設定
[リンクディレクティブ]	リンク・ディレクティブに関する設定
[ROM 化]	ROM 化に関する設定
[フラッシュ]	フラッシュに関する設定
[デバイス]	デバイスに関するオプションの設定

なお，ライブラリ作成のプロジェクトの場合は，次の 4 個のタブになります。

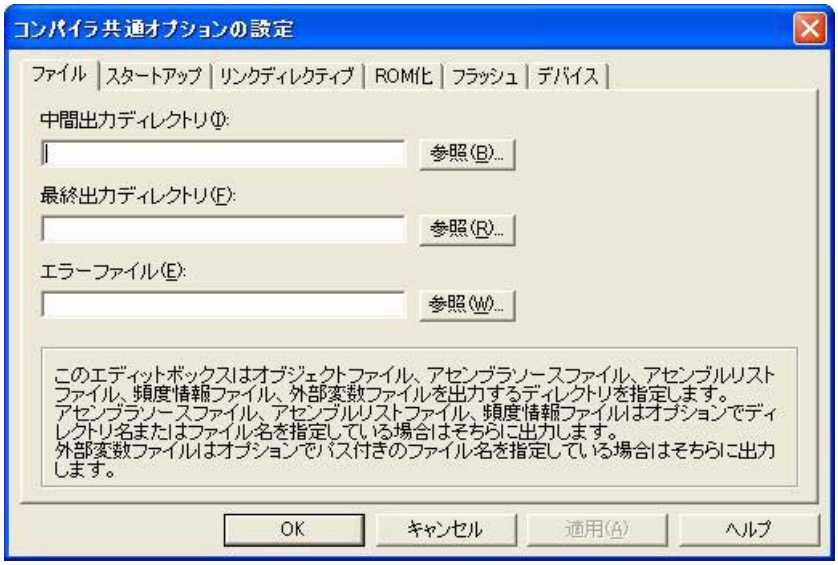
表 3 － 4 [コンパイラ共通オプションの設定] ダイアログ（ライブラリ）

タブ	内容
[ファイル]	出力ファイルに関するの設定
[ROM 化]（ライブラリ）	ライブラリ作成時の ROM 化に関する設定
[フラッシュ]（ライブラリ）	ライブラリ作成時のフラッシュに関する設定
[デバイス]	デバイスに関するオプションの設定

[ファイル]

出力ファイルに関する設定を行います。

図 3 － 2 [コンパイラ共通オプションの設定] ダイアログ：[ファイル] タブ



(1) 中間出力ディレクトリ

中間出力用フォルダを設定するエディット・ボックスです。中間出力用フォルダとは，オブジェクト・ファイル，アセンブリ言語ソース・ファイル，アセンブル・リスト・ファイル，頻度情報ファイル，および外部変数ファイルを出力するフォルダです。

(2) 最終出力ディレクトリ

最終出力用フォルダを設定するエディット・ボックスです。最終出力用フォルダとは，実行可能オブジェクト・ファイル，ヘキサ・ファイル，アーカイブ・ファイル，リンク・マップ・ファイル，エラー・ファイル，およびセクション・ファイルを出力するフォルダです。

ただし，それぞれのオプションでパス付きのファイル名を指定している場合は，そちらに出力します。

(3) エラーファイル

エラーと警告メッセージを出力するファイルを設定するエディット・ボックスです。このエディット・ボックスを設定するとすべてのコンパイラ・ツールに “ +err_file ” オプションを設定します。

[備考]

ダイアログの [OK] **ボタン，または [適用] ボタンを押した際に，“ 中間出力ディレクトリ ”，または “ 最終出力ディレクトリ ” で指定したフォルダが存在しなかった場合は，フォルダの作成を確認します。**

図 3 － 3 フォルダ作成の確認

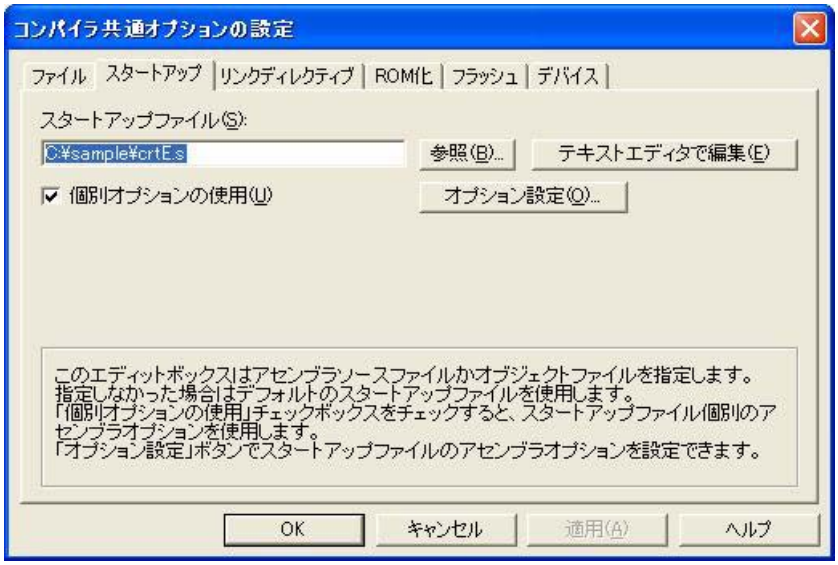
上記のメッセージ・ボックスで [はい] ボタンを押した場合は，フォルダを作成します。

なお，フォルダを作成しない場合は，オプションを設定できません。

[スタートアップ]

スタート・アップ・ファイルに関する設定を行います。

図 3 － 4 [コンパイラ共通オプションの設定] ダイアログ：[スタートアップ] タブ



(1) スタートアップファイル

スタート・アップ・ファイルを設定するエディット・ボックスです。アセンブリ言語ソース・ファイル，またはオブジェクト・ファイルを設定します。スタート・アップ・ファイルを設定しなかった場合は，デフォルト（crtN.o，または crtE.o）を使用します。

アセンブリ言語ソース・ファイルは，[テキストエディタで編集] ボタンで編集することができます。ただし，このエディット・ボックスでは，フォルダ名にスペースは使用できますが，ファイル名にスペースは使用できません。

(2) 個別オプションの使用

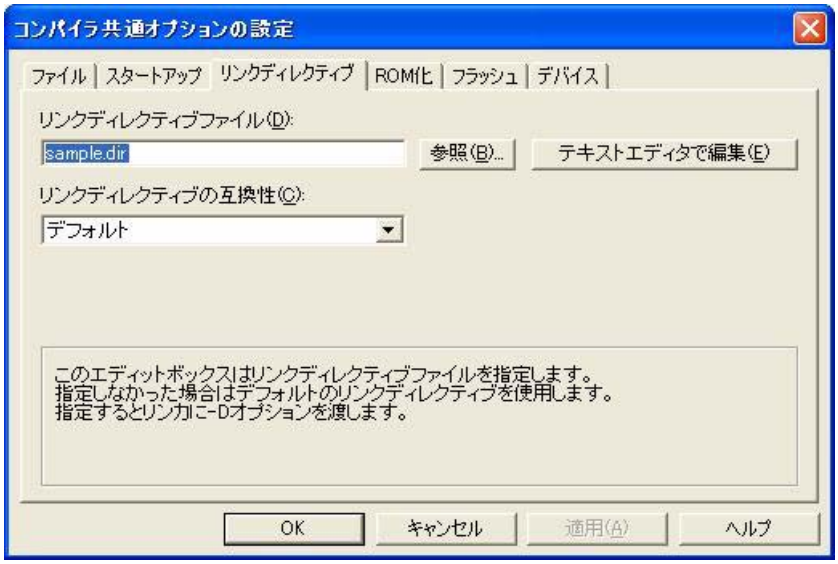
チェックされている場合は，スタート・アップ・ファイル個別のアセンブラ・オプションを使用します。

[オプション設定] ボタンでスタート・アップ・ファイルのアセンブラ・オプションを設定できます。

[リンクディレクティブ]

リンク・ディレクティブに関する設定を行います。

図 3 － 5 [コンパイラ共通オプションの設定] ダイアログ：[リンクディレクティブ] タブ



(1) リンクディレクティブファイル

リンク・ディレクティブ・ファイルを設定するエディット・ボックスです。設定すると，リンカに -D オプションが渡されます。リンク・ディレクティブ・ファイルは，[テキストエディタで編集] ボタンで編集することができます。

なお，このエディット・ボックスでは，フォルダ名にスペースは使用できますが，ファイル名にスペースは使用できません。また，ファイル名に拡張子（ .dir ’ 推奨）を指定しない場合，ビルド処理が正常に行われない可能性を示す警告メッセージが表示されます。

(2) リンクディレクティブの互換性

CA850 Ver.2.70 以前の版とのリンク・ディレクティブの互換性を設定するエディット・ボックスです。

次の 4 つの項目からなります。

- デフォルト
- V2.40 以前
- V2.50
- V2.60

以前の版を指定した場合，リンカに “ -Xolddir ” オプションが渡されます。

注意 “ V2.40 以前 ” には，Ver.2.4x が含まれます。

[ROM 化]

ROM 化に関する設定を行います。

図 3 － 6 [コンパイラ共通オプションの設定] ダイアログ：[ROM 化] タブ

(1) ROM 化用オブジェクトの生成

チェックされている場合は，ROM 化プロセッサに対応したオブジェクトを生成します。指定するとコンパイラに -Xr オプション，リンカに -lr オプションと rompcrt ファイルを渡して，ROM 化プロセッサを使用します。

(2) rompcrt ファイル

ROM 化を行う場合に指定する rompcrt ファイルのファイル名を設定するエディット・ボックスです。指定しなかった場合は，デフォルト（rompcrt.o）を使用します。アセンブリ言語ソース・ファイルは，[テキストエディタで編集] ボタンで編集することができます。

ただし，このエディット・ボックスでは，フォルダ名にスペースは使用できますが，ファイル名にスペースは使用できません。

(3) 個別オプションの使用

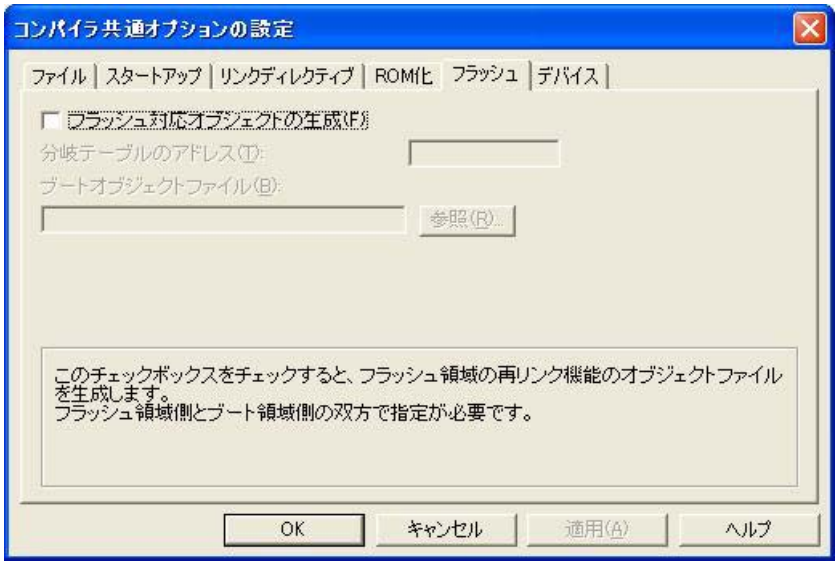
チェックされている場合は，rompcrt ファイル個別のアセンブラ・オプションを使用します。

[オプション設定] ボタンで rompcrt ファイルのアセンブラ・オプションを設定できます。

[フラッシュ]

フラッシュに関する設定を行います。

図 3 － 7 [コンパイラ共通オプションの設定] ダイアログ：[フラッシュ] タブ



(1) フラッシュ対応オブジェクトの生成

チェックされている場合は，フラッシュ領域の再リンク機能のオブジェクト・ファイルを生成します。フラッシュ領域側とブート領域側での指定が必要です。

フラッシュ対応オブジェクトについての詳細は「5.6 フラッシュ／外付け ROM 再リンク機能」を参照してください。

(2) 分岐テーブルのアドレス

分岐テーブルのアドレスを設定するエディット・ボックスです。分岐テーブルを配置するアドレスを C 言語の 16 進数で指定します。フラッシュ領域側とブート領域側の双方で同じアドレスの指定が必要です。指定するとリンカに -ext_table オプションが指定されます。

(3) ブートオブジェクトファイル

ブート・オブジェクト・ファイルを設定するエディット・ボックスです。指定するとコンパイラに -Wa,-zf オプション，アセンブラに -zf オプション，リンカに -zf オプションが渡されます。

ただし，このエディット・ボックスでは，フォルダ名にスペースは使用できますが，ファイル名にスペースは使用できません。

[デバイス]

デバイスに関するオプションの設定を行います。

図 3 － 8 [コンパイラ共通オプションの設定] ダイアログ：[デバイス] タブ

(1) 256M バイトモード

チェックされている場合は，256M バイト・モードを指定します。指定するとコンパイラ，アセンブラ，リンカに -X256M オプションが渡されます。これによりメモリ空間を 256M バイトとして扱います。

このオプションを指定しない場合はメモリ空間を 64M バイトとして扱い，アドレス解決します。このオプションは使用するチップセットにあわせて設定してください。

V850Ex コアでは，物理アドレス空間が 256M バイト持つ場合が多く，64M を越えた 256M までの空間を使用するアプリケーションを作成する場合，このオプションを指定してください。

(2) BPC レジスタ

プログラマブル周辺 I/O レジスタの上位アドレスを指定します。指定するとコンパイラに -Xbpc オプション，as850 に -bpc オプションが渡されます。

ターゲット・デバイスがプログラマブル周辺 I/O レジスタ機能を持ち（V850E/IA1 など），変更可能なアドレス部分（=BPC レジスタに設定する値）を設定したい場合，アプリケーションのコンパイル（アセンブル）時に，値を確定させる必要があります。そこでこのオプションを指定すると，指定した値を使用してコンパイル（アセンブル）します。

このオプション指定時には必ず値を指定してください。値には 2 進数，8 進数，10 進数，16 進数を使用できます。不正な値を指定した場合，および BPC レジスタに設定可能な範囲を越える値を指定した場合，警告メッセージが出力され，このオプションは無視されます。

例

```
-Xbpc=0x1234
```

上記の場合，ターゲット・デバイスが V850E/IA1 の場合，プログラマブル周辺 I/O レジスタ領域の先頭アドレスは，この値を 14 ビット左シフトした 0x48d0000 として扱われます。

設定する値はアプリケーション全体で 1 つです。ファイルごとのオプション設定で“ -Xbpc ”や“ -bpc ”を指定する場合，ファイル間で同じ値にしてください。ただし，プログラマブル周辺 I/O レジスタを使用しないファイルに対しては，このオプションを指定する必要はありません。

プログラマブル周辺 I/O レジスタ機能を持たないターゲット・デバイスの場合，および V850 ／ V850Ex ／ V850E2 コア共通としてアセンブルする場合，このオプションを指定すると，警告メッセージが出力され，このオプションは無視されます。

なお，このオプションは，プログラマブル周辺 I/O レジスタのアドレスをコンパイル（アセンブル）時に確定するためのものであり，BPC レジスタに実際に値を反映させるものではありません。動作のためには，別途，スタート・アップ・モジュールなどで BPC レジスタに値を設定する必要があります。

“ CA850 ユーザーズ・マニュアル C 言語編 ”にスタート・アップ・ルーチンのサンプルが載っていますので，そちらを参照してください。また，パッケージに含まれるスタート・アップ・モジュールにも，サンプルが載っています（コメントアウトしてあります）。

例

V850E/IA1 で，プログラマブル周辺 I/O レジスタの先頭アドレスの変更可能部分を“ 0x1234 ”とし，この機能の使用を許可するフラグ“ 0x8000 ”を設定する場合，次の記述をスタート・アップ・モジュールに記述します。

```
mov    0x9234,r10    -- 0x1234 | 0x8000 = 0x9234
st.h   r10, BPC
```

as850 では，このオプションを指定するか，または省略していても，実際にはプログラマブル周辺 I/O レジスタを参照していた場合，予約セクションである .bpc セクションを出力します。このセクションは，リンク時のチェックのために使用されます。.bpc セクションは，情報用の特殊な予約セクションであり，メモリにロードされることはありません。したがって，通常のセクションのように，リンク・ディレクティブに記述する必要はありません。

(3) セキュリティ ID

フラッシュ・メモリ搭載デバイスの“ セキュリティ ID ”を設定します。セキュリティ ID 機能をサポートしていないデバイスの場合は利用できません。

ID は，10 バイト以内の 16 進数（先頭の 0x も含む）で指定します。

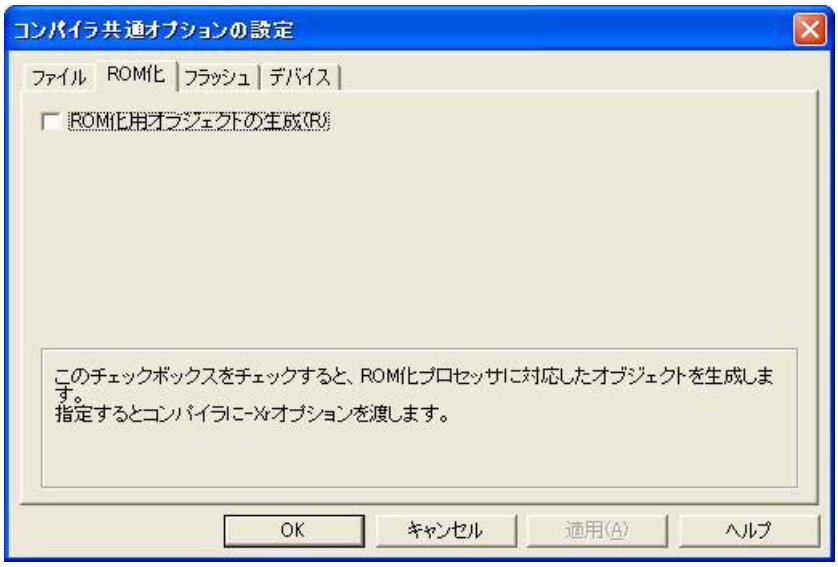
なお，指定した値が 10 バイトより不足している場合は，上位ビットを 0 で埋めます。10 バイトを越えた場合はエラーを出力します。ただし，セキュリティ ID 機能をサポートしていないデバイス用のオブジェクトをリンク時に指定した場合は，警告メッセージを出力し，指定は無視されます。

たとえば，セキュリティ・コード 0x112233445566778899aa を設定する場合は，エディット・ボックスにこの値をそのまま記述します。

[ROM 化]（ライブラリ）

ライブラリ作成時の ROM 化に関する設定を行います。

図 3 － 9 [コンパイラ共通オプションの設定] ダイアログ：[ROM 化]（ライブラリ）タブ



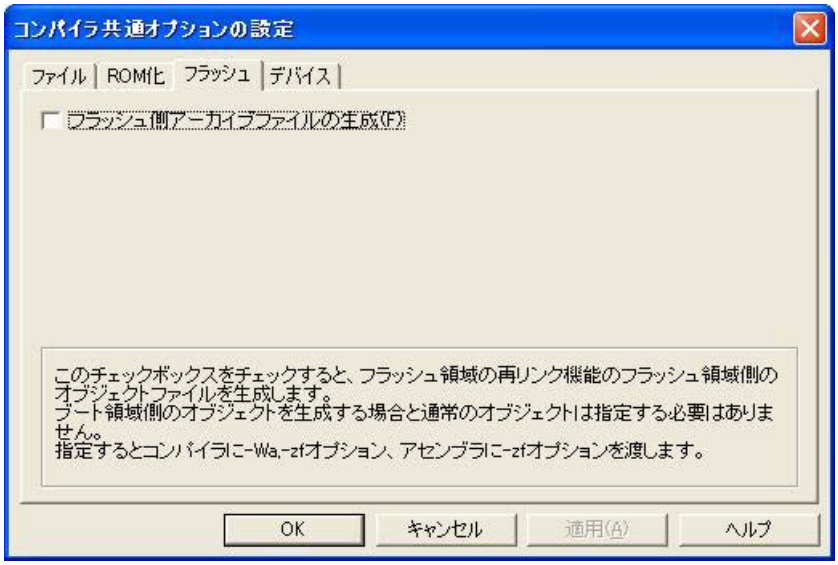
(1) ROM 化用オブジェクトの生成

チェックされている場合は，ROM 化プロセッサに対応したオブジェクトを生成します。指定するとコンパイラに -Xr オプションを渡します。

[フラッシュ]（ライブラリ）

ライブラリ作成時のフラッシュに関する設定を行います。

図 3 － 10 [コンパイラ共通オプションの設定] ダイアログ：[フラッシュ]（ライブラリ）タブ



(1) フラッシュ側アーカイブファイルの生成

チェックされている場合は，フラッシュ領域の再リンク機能のフラッシュ領域側のオブジェクト・ファイルを生成します。ブート領域側のオブジェクトを生成する場合と通常のオブジェクトは，指定する必要はありません。指定するとコンパイラに -Wa,-zf オプション，as850 に -zf オプションが渡されます。

フラッシュ対応オブジェクトについての詳細は「5.6 フラッシュ／外付け ROM 再リンク機能」を参照してください。

3.6.2 [コンパイラオプションの設定] ダイアログ

[コンパイラオプションの設定] ダイアログの上部には，次の 12 個のタブが表示されており，タブの選択により，ダイアログの表示内容が変わります。

表 3 － 5 [コンパイラオプションの設定] ダイアログ

タブ	内容
[一般]	コンパイラの利用頻度の高いオプションの設定
[入力ファイル]	コンパイラへの入力に関するオプションの設定
[プリプロセッサ]	コンパイル前処理に関するオプションの設定
[C 言語と漢字]	C 言語仕様と漢字コードに関するオプションの設定
[最適化とデバッグ情報]	ソース単位の最適化レベル，およびデバッグ情報の有無
[最適化の詳細設定]	フェーズ単位の最適化に関するオプションの設定
[外部変数レジスタ]	外部変数レジスタに関する設定
[出力ファイル]	出力ファイルに関するオプションの設定
[出力コード]	出力コードに関するオプションの設定
[メッセージ]	メッセージに関するオプションの設定
[アセンブラ]	C 言語ソースのアセンブル時に使用するアセンブラのオプションの設定
[その他]	その他の設定

なお，個別の C 言語ソース・ファイルへのオプション指定の場合は，次の 11 個のタブになります。

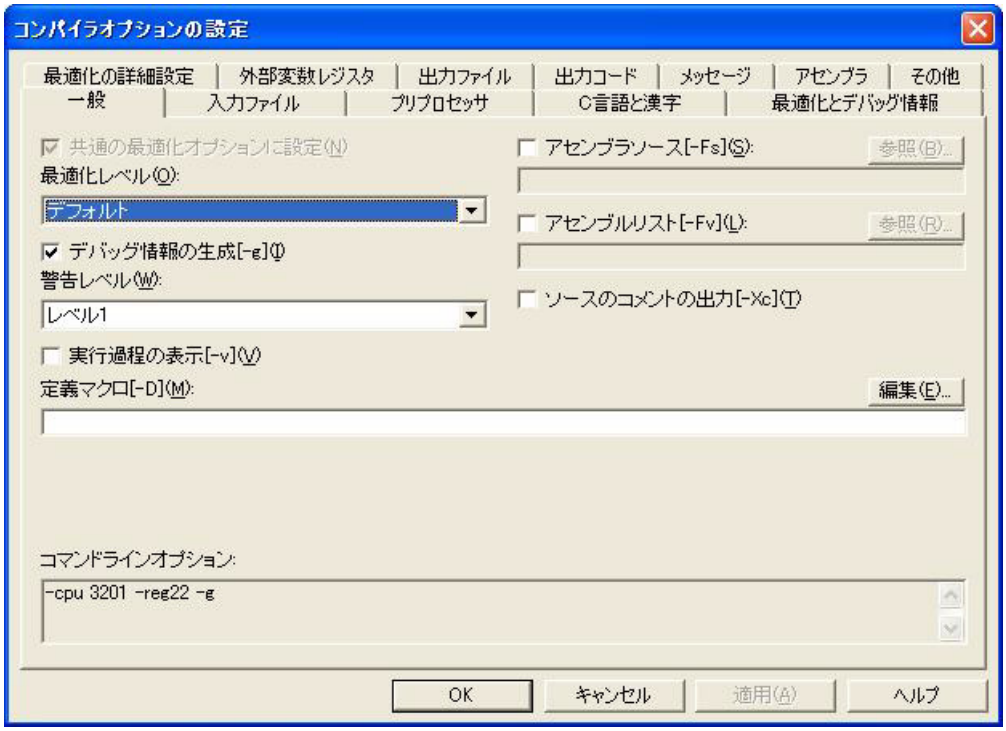
表 3 － 6 [コンパイラオプションの設定] ダイアログ（ソース個別）

タブ	内容
[一般]	コンパイラの利用頻度の高いオプションの設定
[プリプロセッサ]	コンパイル前処理に関するオプションの設定
[C 言語と漢字]	C 言語仕様と漢字コードに関するオプションの設定
[最適化とデバッグ情報]	ソース単位の最適化レベル，およびデバッグ情報の有無
[最適化の詳細設定]	フェーズ単位の最適化に関するオプションの設定
[出力ファイル]	出力ファイルに関するオプションの設定
[出力コード]	出力コードに関するオプションの設定
[メッセージ]	メッセージに関するオプションの設定
[アセンブラ]	C 言語ソースのアセンブル時に使用するアセンブラのオプションの設定
[その他]	その他の設定
[差分]	C 言語ソース全体に対するコンパイラ・オプションとソース個別オプションの差分の表示

[一般]

コンパイラの利用頻度の高いオプションの設定を行います。

図 3 － 11 [コンパイラオプションの設定] ダイアログ：[一般] タブ



(1) 共通の最適化オプションに設定

“ ソース個別オプション ” 設定時に有効となるチェック・ボックスです。ソース個別にかける最適化として，このチェック・ボックスの下の“ 最適化レベル ”で設定されている最適化を使用するか，しないかを設定します。

チェックされている場合は，対象ソースに対しても，全体オプション設定で指定された最適化レベルと同じ最適化を使用することになります。

なお，“ 全体オプション設定 ” 時には，このチェック・ボックスはグレー表示になっており，指定できません。

(2) 最適化レベル

ドロップダウン・リストから，使用する “ 最適化のタイプ ” を選択します。このオプションは，[最適化とデバッグ情報] タブの “ 最適化 ” で設定する項目と同じです。指定できる最適化は，次のいずれかになります。

- デバッグ優先 [-Od]
- デフォルト
- 標準最適化 [-Og]
- 高度な最適化 [-O]
- より高度な最適化（サイズ優先）[-Os]
- より高度な最適化（実行速度優先）[-Ot]

最適化についての詳細は「3.7.3 効率的な最適化の仕方」を参照してください。

(3) デバッグ情報の生成 [-g]

デバッグ情報を生成します。デバッガで C 言語ソース・デバッグを行いたい場合など，プログラムをデバッグする際にはチェックしてください。チェックされている場合は，コンパイラに -g オプションが渡されます。

(4) 警告レベル

ドロップダウン・リストから，出力する警告メッセージレベルを選択します。このオプションは，[メッセージ] タブの “ 警告レベル ” で設定する項目と同じです。

出力する警告メッセージのレベルは次のとおりです。

出力しない [-w]	警告メッセージを抑止する
レベル 1	通常の警告メッセージを出力する（デフォルト）
レベル 2[-w2]	詳細な警告メッセージを出力する

(5) 実行過程の表示 [-v]

ビルド時に ca850 の実行状況の詳細を，アウトプット・ウインドウに表示します。チェックすると，コンパイラ内部の各フェーズの実行状況まで表示します。

このオプションは，[メッセージ] タブの “ 実行過程の表示 [-v] ” で設定する項目と同じです。

(6) アセンブラソース [-Fs]

C 言語ソースのコンパイル結果のアセンブリ言語ソースを出力するかどうかを設定するチェック・ボックスです。チェックされている場合は，下のエディット・ボックスにフォルダ名やファイル名が指定可能になります。このエディット・ボックスに “ 全体オプションの設定 ” として “ フォルダ名 ” を指定し，そのフォルダが存在しなかった場合は，フォルダを作成するかどうかを確認するメッセージ・ボックスが表示されます。エディット・ボックスに何も指定しなかった場合は，C 言語ソース・ファイル名の拡張子を .s として，プロジェクト・フォルダにアセンブリ言語ソースを出力します。別の出力先を指定したい場合は，エディット・ボックスにフォルダ名を指定してください。

ファイル名を指定する場合ですが “ 全体オプションの設定 ” でファイル名を指定すると，同じファイル名で上書きするため，結果的に最後にコンパイルしたソースに対するアセンブリ言語ソースが出力されます。ファイル名を指定するときは “ ソース個別オプション設定 ” でファイル名を指定する場合に有効になります。

このオプションは，[出力ファイル] タブの “ アセンブラソース [-Fs] ” で設定する項目と同じです。

(7) アセンブルリスト [-Fv]

C 言語ソースのコンパイル結果のアセンブル・リストを出力するかどうかを設定するチェック・ボックスです。チェックされている場合は，下のエディット・ボックスにフォルダ名やファイル名が指定可能になります。このエディット・ボックスに “ 全体オプションの設定 ” として “ フォルダ名 ” を指定し，そのフォルダが存在しなかった場合は，フォルダを作成するかどうかを確認するメッセージ・ボックスが表示されます。エディット・ボックスに何も指定しなかった場合は，C 言語ソース・ファイル名の拡張子を .v として，プロジェクト・フォルダにアセンブル・リストを出力します。別の出力先を指定したい場合は，エディット・ボックスにフォルダ名を指定してください。

ファイル名を指定する場合ですが “ 全体オプションの設定 ” でファイル名を指定すると，同じファイル名で上書きするため，結果的に最後にコンパイルしたソースに対するアセンブル・リストが出力されます。ファイル名を指定するときは，“ ソース個別オプション設定 ” でファイル名を指定する場合に有効になります。

このオプションは，[出力ファイル] タブの “ アセンブルリスト [-Fv] ” で設定する項目と同じです。

(8) ソースのコメントの出力 [-Xc]

出力するアセンブリ言語ソース・ファイル，およびアセンブル・リスト中に C 言語ソース・プログラムをコメントとして出力します。“ アセンブラソースの出力 ”，または “ アセンブルリストの出力 ” 指定時にのみ指定可能です。

このオプションは [出力コード] タブの “ ソースのコメントの出力 [-Xc] ” で設定する項目と同じです。

(9) 定義マクロ [-D]

定義したいマクロ名を “ マクロ名 name= 定義値 def ” の形式で指定します。“ = 定義値 def ” の指定を省略した場合，def を 1 とみなします。

例

```
test = 2      /* マクロ “ test ” が “ 2 ” で定義される */
```

C 言語ソース・プログラムの前に #define name def が記述されたものとみなします。複数のマクロを定義する場合，“ ; （セミコロン） ” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，定義マクロ項目をダイアログ上で編集することができます。

なお，マクロ名に空白は使用できません。

このオプションは [プリプロセッサ] タブの “ 定義マクロ [-D] ” で設定する項目と同じです。

(10) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[ボタン]

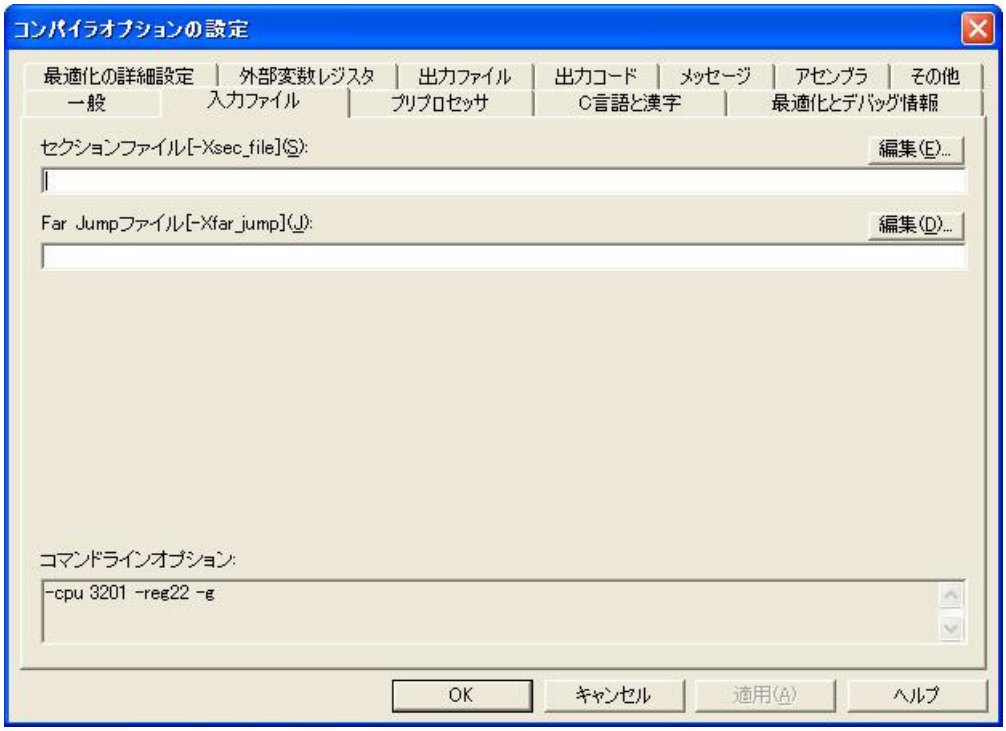
(1) [ソースオプションの削除] ボタン

ソース・ファイル個別のオプション設定を行っている場合に選択可能なボタンです。プロジェクト全体のオプション指定時には表示されず，選択できません。個別オプションを無効にし，全体のオプション設定を有効にしたい場合は，このボタンをクリックします。

[入力ファイル]

コンパイラの入力に関するオプションの設定を行います。

図 3 － 12 [コンパイラオプションの設定] ダイアログ：[入力ファイル] タブ



(1) セクションファイル [-Xsec_file]

セクション・ファイル名を指定します（「第 9 章 セクション・ファイル・ジェネレータ」参照）。テキスト・ボックスにファイル名が設定できます。フォルダ名にスペースは使用できますが，ファイル名にスペースは使用できません。セクション・ファイルを複数に分割しており，このオプションで複数のファイル名を指定する場合，“ ; （セミコロン）” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，ファイル名項目をダイアログ上で編集することができます。

(2) Far Jump ファイル [-Xfar_jump]

-Xfar_jump オプションで指定する Far Jump ファイルを設定します。Far Jump ファイルは，ファイルに記述された関数の分岐命令に対して jmp 命令を使用したコードを出力します。関数本体が，jarl，および jr 命令では分岐できない範囲（± 1M バイト以上）にあり，ld850 がエラーを出力する場合，このオプションを用いてコンパイルし直します。

なお，ファイル名には拡張子が必要となります。推奨する拡張子は “ .fjp ” です。フォルダ名にスペースは使用できますが，ファイル名にスペースは使用できません。[編集] ボタンの選択により表示される [オプションの編集] ダイアログにて，ファイルを選択することができます。

(3) コマンドラインオプション

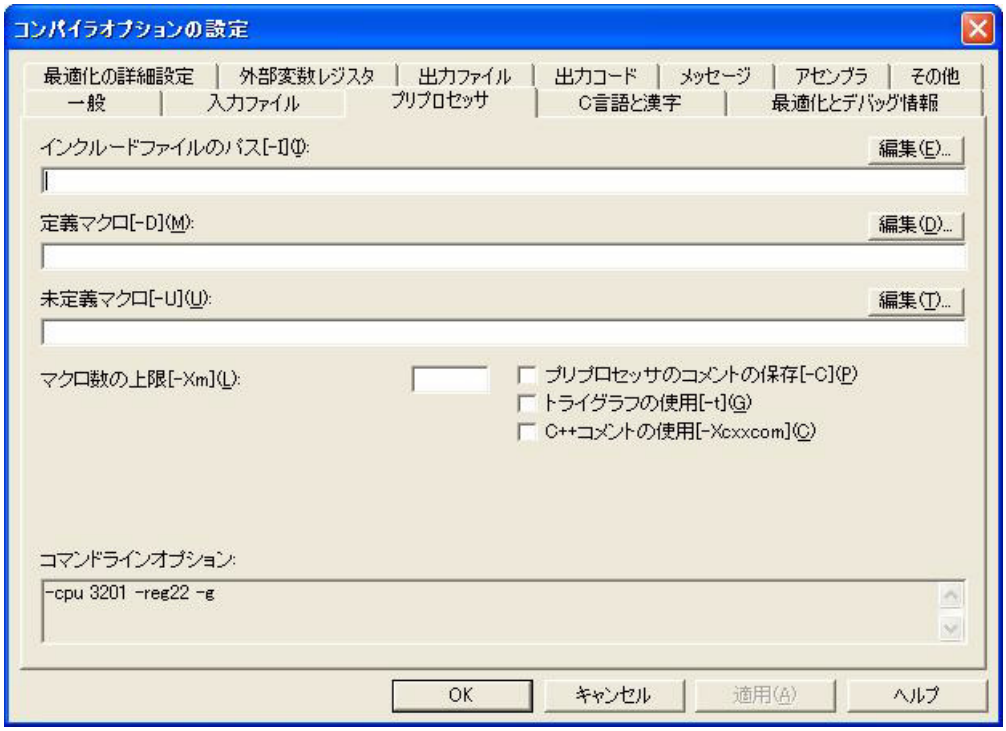
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[プリプロセッサ]

プリプロセッサに関するオプションの設定を行います。

図 3 － 13 [コンパイラオプションの設定] ダイアログ：[プリプロセッサ] タブ



(1) インクルードファイルのパス [-I]

ヘッダ・ファイルを検索するフォルダを指定します。ここで指定したフォルダは，標準フォルダより優先して検索されます。複数のパスを指定する場合，“ ; （セミコロン）” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，パス項目をダイアログ上で編集することができます。

例

```
.\include;C:\include
```

このオプションを省略した場合，標準フォルダ[注1] のみ検索します[注2]。

なお，このオプションは，ソース個別の指定では選択できません。常に全体のオプションとして設定します。

注 1　インストール・フォルダ \inc850 です。

注 2　‘ " ’ によりファイル名を囲む形式では，ソース・ファイルのあるフォルダを最初に検索します。

(2) 定義マクロ [-D]

定義したいマクロ名を “ マクロ名 name= 定義値 def ” の形式で指定します。“ = 定義値 def ” の指定を省略した場合，def を 1 とみなします。

例

```
test = 2      /* マクロ “test” が “2” で定義される */
```

C 言語ソース・プログラムの前に #define name def が記述されたものとみなします。複数のマクロを定義する場合，“ ; （セミコロン） ” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，定義マクロ項目をダイアログ上で編集することができます。

なお，マクロ名に空白は使用できません。

(3) 未定義マクロ [-U]

無効化したいマクロ名 name を指定します。C 言語ソース・プログラムの前に，#undef name が記述されたものとみなします。複数のマクロを無効化する場合，“ ; （セミコロン） ” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，未定義マクロ名をダイアログ上で編集することができます。

例

```
__3201__      /* マクロ “__3201__” が無効化される */
```

なお，マクロ名に空白は使用できません。

(4) マクロ数の上限 [-Xm]

マクロ識別子数の上限を，32,767 までの 10 進数で指定します。デフォルトは 2,047 です。

なお，このオプションはプリプロセッサで使用するバッファのサイズを大きくするものです。

ただし，これによって何文字分のバッファが確保されるのかという具体的な数値を出すことはできません。

(5) プリプロセッサのコメントの保存 [-C]

C 言語ソース・プログラムの前処理の出力に，ソース・プログラムのコメントも含めます。[出力ファイル] タブ の “ プリプロセス処理したソース ” がチェックされている 場合に有効です。

(6) トライグラフの使用 [-t]

トライグラフ系列[注]を置換します。

注 ANSI 規格で規定された，単一文字に置換される 3 文字（トライグラフ）系列。
ANSI 規格に関する文献を参照してください。

(7) C++ コメントの使用 [-Xcxxcom]

通常のコメントのほかに，“ // ” から行末までをコメント（C++ のコメント・スタイル）として扱います。

(8) コマンドラインオプション

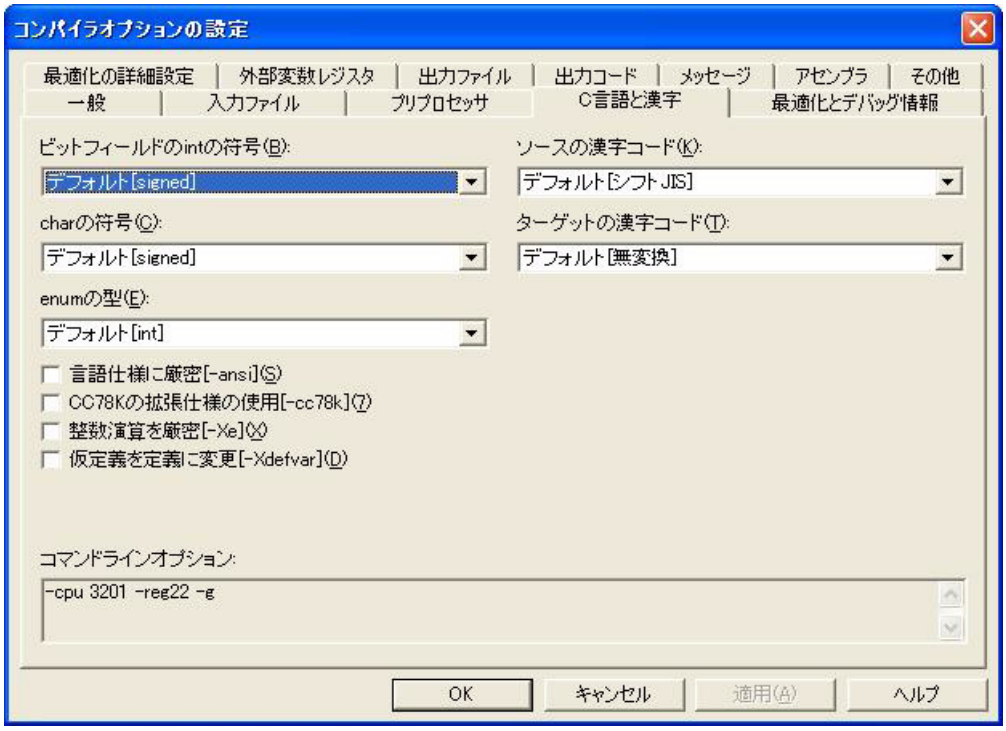
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[C 言語と漢字]

C 言語仕様と漢字コードに関するオプションの設定を行います。

図 3 － 14 [コンパイラオプションの設定] ダイアログ：[C 言語と漢字] タブ



(1) ビットフィールドの int の符号

型指定子（signed，unsigned）の付かない単なる int 型のビット・フィールドに対し，符号付きとするか符号なしとするかを指定します。デフォルトでは，符号付きとして扱います。

(2) char の符号

型指定子（signed，unsigned）の付かない単なる char 型に対し，符号付きとするか符号なしとするかを指定します。デフォルトでは，符号付きとして扱います。

(3) enum の型

列挙型に対し，どの整数型と整合するかを指定します。char，unsigned char，short，unsigned short，int が選択でき，デフォルトでは，int として扱います。

(4) 言語仕様に厳密 [-ansi]

ANSI 規格の言語仕様に厳密なコンパイルを行います。

(5) CC78K の拡張仕様の使用 [-cc78k]

CC78Kx の拡張言語仕様を有効にするかどうかを設定します。チェックされている場合は，-cc78k オプションを指定したのと同じになります。

(6) 整数演算を厳密 [-Xe]

16 ビット・データ以下の整数に対し mulh，divh 命令を利用せず，ランタイム・ライブラリ[注] ___mul ／ ___mulu，___div ／ ___divu を利用します。実行速度は遅くなりますが，ANSI 規格に厳密な乗除算処理を行います。選択しない場合，mulh，divh 命令を利用します。

注 CA850 におけるランタイム・ライブラリは，V850 マイクロコントローラのアーキテクチャにはない命令について，ANSI 規格を満たすために，ca850 の標準ライブラリで用意されているものです。
ランタイム・ライブラリの詳細は " CA850 ユーザーズ・マニュアル C 言語編 " を参照してください。

(7) 仮定義を定義に変更 [-Xdefvar]

変数の仮定義を定義として扱います。

(8) ソースの漢字コード

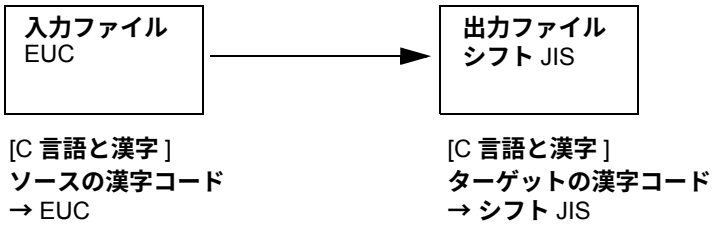
入力ファイル中の日本語のコメント，文字列に対し，使用する文字コードを指定します。入力ファイル中に，指定したコード以外のコードが存在する場合，エラー・メッセージを出力します。" なし "を指定した場合，コードを保証しません。" デフォルト " はシフト JIS となります。

(9) ターゲットの漢字コード

日本語の文字列を，指定されたコードに変換して出力します[注]。アプリケーション開発時に使用した漢字コードを，ターゲットで変更したい場合に有効です。" なし " を指定した場合，および " デフォルト " を指定した場合，コードを変換しません。

なお，" ソースのコメントの出力 " 指定により出力されるコメントは，このオプションに関わらず，変更しません。

注 " ターゲットの漢字コード "指定によりコードを変換したい入力ファイル中で，デフォルト（シフト JIS）以外のコードを使用している場合，入力ファイル・オプションの " ソースの漢字コード " により，使用しているコードを指定してください。
指定しないと，入力はデフォルトのシフト JIS とみなされ，コードが変換されません。



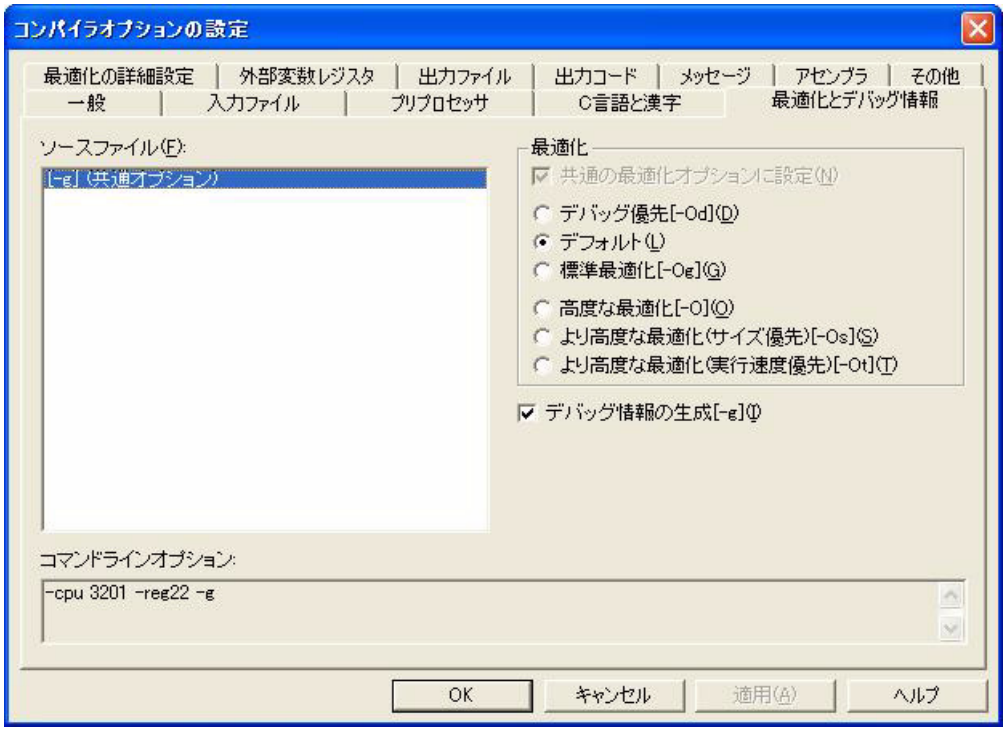
(10) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。
なお，このエリアは参照用のため，書き込みはできません。

[最適化とデバッグ情報]

ソース単位の最適化レベル，およびデバッグ情報の有無の設定を行います。

図 3 － 15 [コンパイラオプションの設定] ダイアログ：[最適化とデバッグ情報] タブ



(1) ソースファイル

現在プロジェクトに登録されているソース・ファイルの一覧です。リスト中のソース・ファイルを選択すると，ソース個別に設定されている「最適化オプション」や「デバッグ情報の有無」が表示されます。

リスト中の（共通オプション）を選択すると，全体にかかるオプションの設定状態が表示されます。このときは，“ 共通の最適化オプションに設定 ” のチェック・ボックスは無効になります。

ソース・ファイルを選択すると，「最適化オプション」の場合，全体にかかるオプションと同じ設定のときは “ 共通の最適化オプションに設定 ” がチェックされています。違う設定のときは，現在設定されている最適化レベルがチェックされています。“ デバッグ情報の生成 [-g] ” の場合，生成する設定になっている場合は，チェックされています。

(2) 最適化

(a) 共通の最適化オプションに設定

下の最適化レベルを使用しないで，全体にかかるオプションの設定を使用するときに，チェックされます。チェックされている場合は，最適化レベルが無効になります。

- デバッグ優先 [-Od]

  ROM 容量や実行速度に着目せず，論理デバッグに着眼したコードを生成します。CA850 Ver.2.41 以前のデフォルト最適化に相当する機能です。
- デフォルト・オプション

  論理デバッグに着眼したコードを生成します。論理デバッグに影響のない範囲で最適化を行います。
- 標準最適化 [-Og]

  適度な最適化を行います。ほとんどのケースで C 言語ソース・デバッグが可能となる最適化を行います。外部変数をレジスタに割り当てることから，実行速度，コード・サイズともにデフォルト・オプションより改善されます。
- 高度な最適化 [-O]

  ROM 容量に着目した最適化を行います。
- より高度な最適化（オブジェクト・サイズ優先）[-Os]

  ROM 容量を重視した最大限の最適化を行います。
- より高度な最適化（実行速度優先）[-Ot]

  ROM 容量よりも実行速度を重視した最大限の最適化を行います。

(3) デバッグ情報の生成 [-g]

チェックすると，デバッグ情報を生成します。デバッガで C 言語ソース・デバッグを行いたい場合など，プログラムをデバッグする際にはチェックしてください。ソース個別にデバッグ情報を出力させるか否かを設定することができます。

(4) コマンドラインオプション

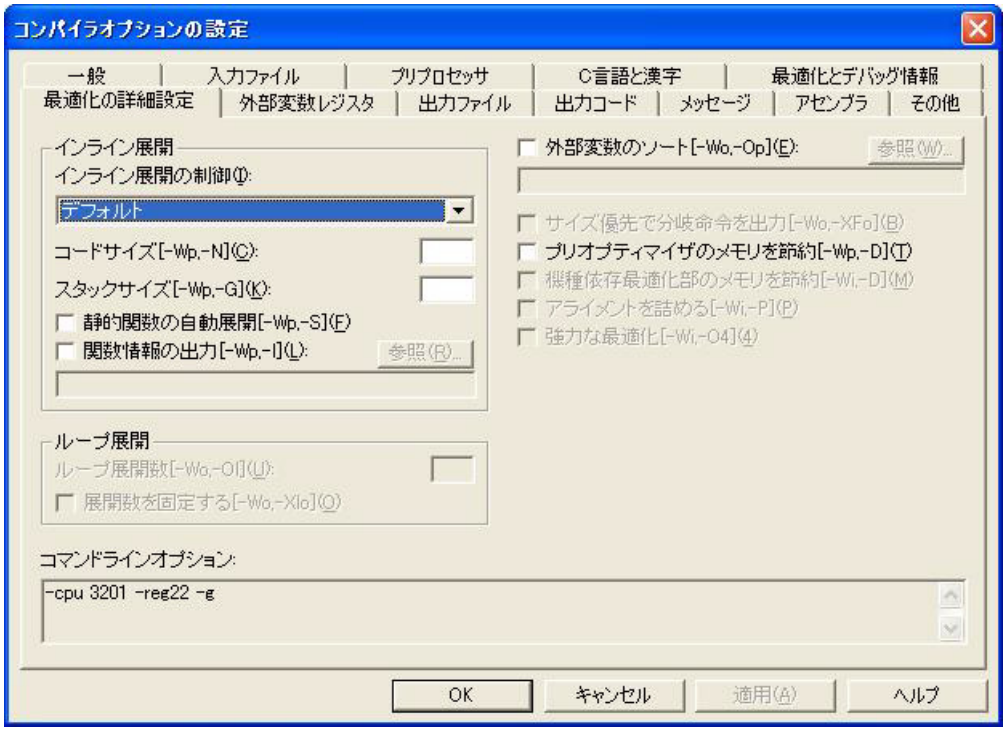
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[最適化の詳細設定]

フェーズ単位の最適化に関するオプションの設定を行います。

図 3 － 16 [コンパイラオプションの設定] ダイアログ：[最適化の詳細設定] タブ



(1) インライン展開

(a) インライン展開の制御

プリオプティマイザの -inline，および -no_inline オプションで指定するインライン展開全体の制御を設定します。

(b) コードサイズ [-Wp,-N]

インライン展開対象関数の中間言語サイズを指定バイトに制限し，それより大きいものはインライン展開しません。サイズの目安については，“ 関数情報の出力 ” を参照してください。デフォルトでは，128 を指定します。

(c) スタックサイズ [-Wp,-G]

インライン展開対象関数の中間言語でのスタック・サイズを指定バイトに制限し，それより大きいものはインライン展開しません。サイズの目安については，“ 関数情報の出力 ” で表示される情報を参照してください。デフォルトでは 32 を指定します。

(d) 静的関数の自動展開 [-Wp,-S]

一度しか参照されない静的な関数をインライン展開します。

(e) **関数情報の出力** [-Wp,-l]

関数の情報を表示します。表示される情報は，“ コード・サイズ ”, “ スタック・サイズ ” の上限を指定する目安になります。

エディット・ボックスに指定しない場合は，標準出力に出力します。

エディット・ボックスにファイル名を指定した場合は指定したファイルに出力します。

図 3 － 17 関数情報の出力（関数名 *func*）

```
-------------------------------- ------ -----
function name                     code  stack
-------------------------------- ------ -----
_func                                7     8
-------------------------------- ------ -----
```

なお，このオプションによって出力されるスタック・サイズは，あくまでもプリオプティマイザが出力する中間言語でのサイズであるため，関数が実際に使用するスタック・サイズとは異なります。ご注意ください。

(2) ループ展開

(a) **ループ展開数** [-Wo,-Ol]

for，while **などで，ループを** *num* **回展開します。より高度な最適化（実行速度優先）の場合にのみ指定できます。実行回数が** N **回（**N **は定数）のループの実行と** *num* **回展開されたコードを含むループの実行に変換されます。**

ただし，展開後のコード・サイズが大きいかループの実行回数が少ない場合は，展開数が少なくなったり展開されない場合があります。また，内側にループを含むような複雑な構造のループは，展開されない場合があります。 *num* **に** 0**，または** 1 **を指定した場合，展開が抑止されます。より高度な最適化（実行速度優先）指定時で，ループ展開は行いたくない場合に有効です。また，** *num* **を指定しない場合，**4 **が指定されたものとみなします。なお，** *num* **は** 10 **進数で指定してください。**

(b) **展開数を固定する** [-Wo,-Xlo]

広域最適部の -Xlo **オプションで指定するループ展開を以前のバージョンと同様な展開をするかどうかを指定するチェック・ボックスです。このチェック・ボックスは，より高度な最適化（実行速度優先）指定時のみ有効です。チェックされている場合は，**-Wo,-Xlo **オプションを指定したのと同じになります。**

(3) 外部変数のソート [-Wo,-Op]

広域最適部の -Op **オプションで指定する外部変数のソートをするかどうかを設定します。チェックされている場合は，オプションの指定による外部変数のソートを行います。エディット・ボックスに指定しない場合は，ファイル内でソートを行います。エディット・ボックスにファイル名を指定した場合は外部変数ファイルを利用したソートを行います。**

指定したファイルは，すべてのソースのコンパイル終了後にコンパイルを行いリンクします。エディット・ボックスのファイル名には，ソース・ファイルと同じベースネーム（拡張子を除いたファイル名）を持つファイル名，または拡張子が .ic **でないファイル名を指定することはできません。**

なお，-Wo,-Op **オプションは，コマンド・ライン・オプションには，表示されません。**

(4) サイズ優先で分岐を出力 [-Wo,-XFo]

広域最適化部の -XFo オプションで指定する分岐命令をコード・サイズ優先で出力するかどうかを指定します。-Og ／ -O ／ -Os ／ -Ot 指定時にのみ使用できます。

(5) プリオプティマイザのメモリを節約 [-Wp,-D]

コンパイル時のプリオプティマイザのメモリ使用量を減らします。マシンのメモリが不足しコンパイルが正常に終了しないときにこのオプションを指定します。このオプションを指定するとコンパイル速度は低下します。

(6) 機種依存最適化部のメモリを節約 [-Wi,-D]

コンパイル時の機種依存最適化部のメモリ使用量を減らします。マシンのメモリが不足しコンパイルが正常に終了しないときにこのオプションを指定します。このオプションを指定するとコンパイル速度は低下します。

(7) アライメントを詰める [-Wi,-P]

分岐先ラベルを整列する最適化を抑止します。これにより，実行コードのサイズを小さくできます。このオプションは，高度な最適化，より高度な最適化（実行速度優先）-Ot オプションを指定しているときに有効です。

また，より高度な最適化（サイズ優先）時には，このオプション機能が含まれるため淡色表示となり，指定できません。

(8) 強力な最適化 [-Wi,-O4]

データ・フロー解析を厳密に行い，最も強い最適化を行います。最適化オプション -O ／ -Os ／ -Ot を指定した上で，さらに強い最適化を行いたい場合に指定してください。

この最適化では具体的に次のことを行います。

- 分岐命令をまたいだレジスタの最適化
- 絶対値演算の最適化
- 分岐命令をまたいだ cmp 命令の最適化
- 分岐命令をまたいだ復帰命令の最適化

なお，ソースによっては -Os ／ -Ot の最適化結果と同じになることがあります。また，コンパイル時間は，-Os ／ -Ot 指定時より遅くなります。

(9) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[備考]

(a) **外部変数のソート**

-Wo,-Op **オプションを用いることによって，外部変数をアライメントが大きい順に並び替えることができます。ただし，**const ／ sconst **セクションに配置されている変数は並び替えません。**

このオプションは -Wo,-Op=*file* **のようにファイル名を指定可能です。**

ファイル名を指定した場合の動作は，次のとおりです。

- **指定した中間言語ファイル** *file* **を読み込んで外部変数のリストを作成します。指定したファイルが存在しない場合は空のリストを作成して次に進みます。**
- **ソース・ファイルの中に外部変数の定義，または仮定義が存在した場合は，上記で作成したリストに移動します。移動の際に仮定義は定義に変更されます。つまり，ソース・ファイルの定義（仮定義）は宣言に置き換えられます。リストに同名の変数定義がすでに存在する場合は，ソース・ファイルの定義に置き換えます。**

  置き換え時にセクション，サイズ，アライメント，初期値のいずれかに変更があった場合は警告メッセージを出力します。
- **外部変数のリストを中間言語ファイル** *file* **に出力します。最後にコンパイルしたファイル全部と** *file* **をコンパイルしたファイルをリンクします。**

  上記のファイル名を指定した場合は，次の注意点があります。

  (1) **オプションを指定してコンパイルしたファイルで特定の変数がすべて仮定義で，オプションを指定しなかったファイルに定義が存在する場合は，オプションを指定しなかった場合は正常にリンクできますが，オプションを指定した場合は多重定義でリンク・エラーになります。**

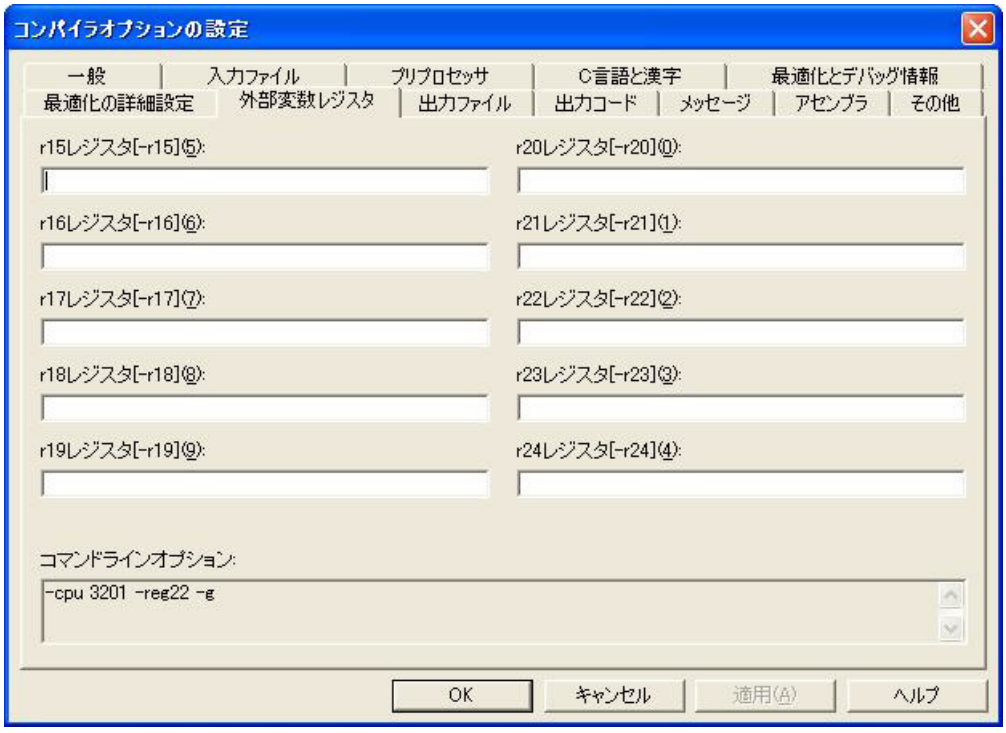
  (2) **オプションを指定してコンパイルしたファイルで特定の変数が多重定義の場合は，リンク・エラーにならないで後でコンパイルしたファイルの定義内容が有効になります。**

  (3) **中間言語ファイル内の変数を削除したい場合は，中間言語ファイル自体を削除してリビルドしてください。**

[外部変数レジスタ]

外部変数レジスタに関する設定を行います。

図 3 － 18 [コンパイラオプションの設定] ダイアログ：[外部変数レジスタ] タブ



(1) r*num* レジスタ [-r*num*]

r*num* オプションで指定するレジスタに割り付ける外部変数を指定します。*num* は，15 から 24 までの 10 個あります。指定されている場合は，-r*num* オプションを指定したのと同じになります。

(2) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[備考]

(a) **外部変数レジスタ**

-r*XY* **オプションを用いることにより，外部変数をレジスタに割り付けることができます。レジスタは** -reg **オプションを指定して空けたマスク・レジスタ以外のレジスタを指定します。外部変数は，‘ _ ’ を除くシンボル名で指定します。**

なお，次の外部変数は指定できません。

- volatile **変数**
- **アドレス演算子 “** & **” を使用した変数**
- **構造体**
- **配列**
- **内部結合する変数（**static**）**
- **周辺** I/O **レジスタ**

指定した外部変数の定義（仮定義）と宣言は削除されます。

外部変数の初期値を利用する（プログラム実行時の最初に初期化しない場合）は，スタート・アップ・ファイルでレジスタに初期値を代入してください。

例

```
int    i = 1;
```

上記のように定義したソース・ファイルで “ -reg26 -r19=i **” と指定した場合は，次のようになります。**

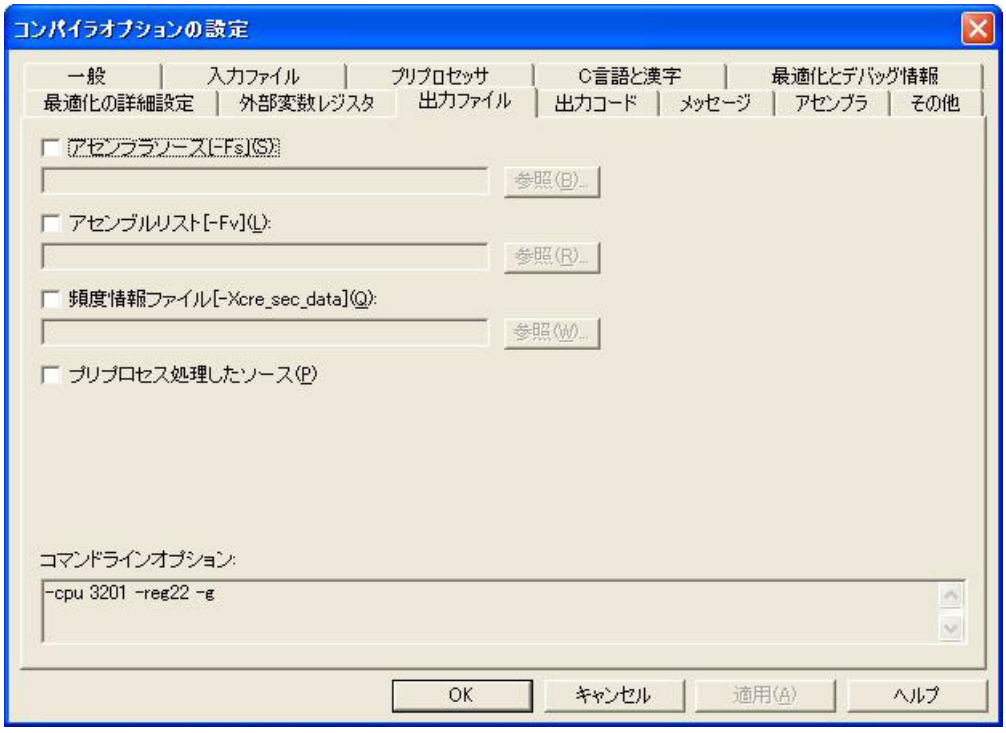
```
mov    1, r19              -- set i
```

注　指定した外部変数のデバッグ情報も削除されるため，最適化デバッグはできません（レジスタ・ウインドウを使用します）。

[出力ファイル]

出力ファイルに関するオプションの設定を行います。

図 3 － 19 [コンパイラオプションの設定] ダイアログ：[出力ファイル] タブ



(1) アセンブラソース [-Fs]

C 言語ソースのコンパイル結果のアセンブリ言語ソースを出力するかどうかを設定するチェック・ボックスです。チェックされている場合は，下のエディット・ボックスにフォルダ名やファイル名が指定可能になります。このエディット・ボックスに“ 全体オプションの設定 ”として“ フォルダ名 ”を指定し，そのフォルダが存在しなかった場合は，フォルダを作成するかどうかを確認するメッセージ・ボックスが表示されます。エディット・ボックスに何も指定しなかった場合は，C 言語ソース・ファイル名の拡張子を .s として，プロジェクト・フォルダにアセンブリ言語ソースを出力します。別の出力先を指定したい場合は，エディット・ボックスにフォルダ名を指定してください。

ファイル名を指定する場合，“ 全体オプション設定 ”でファイル名を指定すると，同じファイル名で上書きするため，結果的に最後にコンパイルしたソースに対するアセンブリ言語ソースが出力されます。ファイル名を指定するときは，“ ソース個別オプション設定 ”でファイル名を指定する場合に有効になります。

(2) アセンブルリスト [-Fv]

C 言語ソースのコンパイル結果のアセンブル・リストを出力するかどうかを設定するチェック・ボックスです。チェックされている場合は，下のエディット・ボックスにフォルダ名やファイル名が指定可能になります。このエディット・ボックスに“全体オプション設定”として“フォルダ名”を指定し，そのフォルダが存在しなかった場合は，フォルダを作成するかどうかを確認するメッセージ・ボックスが表示されます。エディット・ボックスに何も指定しなかった場合は，C 言語ソース・ファイル名の拡張子を .v として，プロジェクト・フォルダにアセンブル・リストを出力します。別の出力先を指定したい場合は，エディット・ボックスにフォルダ名を指定してください。

ファイル名を指定する場合ですが“全体オプション設定”でファイル名を指定すると，同じファイル名で上書きするため，結果的に最後にコンパイルしたソースに対するアセンブル・リストが出力されます。ファイル名を指定するときは，“ソース個別オプション設定”でファイル名を指定する場合に有効になります。

(3) 頻度情報ファイル [-Xcre_sec_data]

セクション・ファイル・ジェネレータで使用する変数の頻度情報ファイルを出力するかどうかを設定するチェック・ボックスです。チェックされている場合は，下のエディット・ボックスにフォルダ名やファイル名が指定可能になります。このエディット・ボックスに " 全体オプションの設定 " として“フォルダ名”を指定し，そのフォルダが存在しなかった場合は，フォルダを作成するかどうかを確認するメッセージ・ボックスが表示されます。エディット・ボックスに何も指定しなかった場合は，C 言語ソース・ファイル名の拡張子を .sec として，プロジェクト・フォルダに頻度情報ファイルを出力します。別の出力先を指定したい場合は，エディット・ボックスにフォルダ名を指定してください。

ファイル名を指定する場合，“全体オプション設定”でファイル名を指定すると，同じファイル名で上書きするため，結果的に最後にコンパイルしたソースに対するアセンブリ言語ソースが出力されます。つまり，複数 C 言語ソース・ファイルでそれぞれに頻度情報ファイル名を指定する場合，“ソース個別オプション設定”，でファイル名を指定してください。

なお，このオプションは C 言語ソース・ファイル内の変数に対する頻度情報ファイルを出力しますが，アセンブリ言語ソース・ファイルに対しては出力しません。

(4) プリプロセス処理したソース

C 言語ソース・プログラムに対し前処理のみ実行して，結果を C 言語ソース・ファイル名の .c を .i に置き換えた名前のファイルに出力します。ソース・プログラムの行番号表示やファイル名表示は出力しません。

(5) コマンドラインオプション

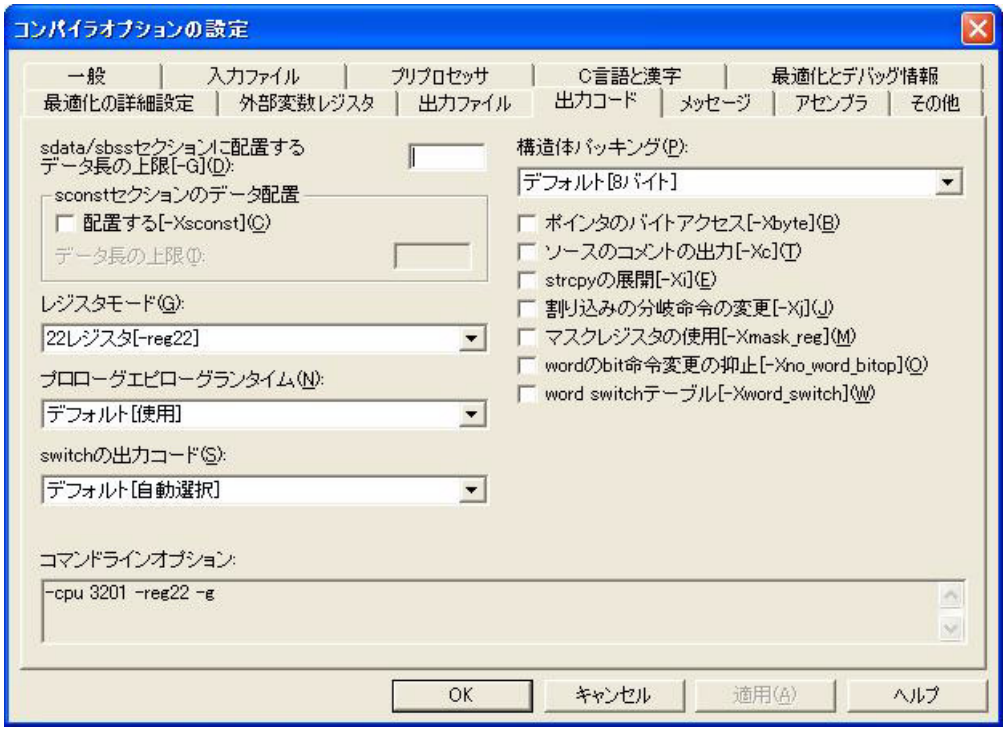
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[出力コード]

出力コードに関するオプションの設定を行います。

図 3 － 20 [コンパイラオプションの設定] ダイアログ：[出力コード] タブ



(1) sdata ／ sbss セクションに配置するデータ長の上限 [-G]

.sdata ／ .sbss セクションに割り当てるデータ長の上限を指定します。指定したサイズ（バイト）以下のデータを .sdata セクション，または .sbss セクションに割り付けます。

ただし，#pragma section 指令やセクション・ファイルで .sdata ／ .sbss セクションが指定されたデータは，そのサイズに関係なく，.sdata ／ .sbss セクションに割り付けます。

指定できる値は 10 進数で 0 ～ 32,767 の範囲が指定可能です。ここで指定する値の目安は，[リンカオプションの設定] ダイアログの [オプション] タブの“ GP 情報の表示 [-A] ”を指定することで出力されるので，参考にしてください。なお，このオプションは，ソース個別の指定では選択できません。常に全体のオプションとして設定します。

(2) sconst セクションのデータ配置

const 属性，つまり，型修飾子 const の付けたデータ，文字列リテラルを sconst セクションに配置する設定を行います。

(a) 配置する [-Xsconst]

チェックすることにより，const 属性のデータ，文字列リテラルを sconst セクションに配置します。

(b) データ長の上限

データ長の上限を入力すると,指定したサイズ(バイト)以下のデータを .sconst セクションに割り付けます。ただし，#pragma section 指令やセクション・ファイルで .sconst セクションが指定されたデータはそのサイズに関係なく .sconst セクションに割り付けます。何も入力しないと，const 属性のデータ，文字列リテラルを .sconst セクションに割り付けます。つまり，指定できる値は 10 進数で 0 ～ 32,767 の範囲の整数が指定可能です。

.sconst セクション，.const セクションについての詳細は“ CA850 ユーザーズ・マニュアル C 言語編 ”を参照してください。

注意 #pragma section 指令やセクション・ファイルでセクションを指定せず，かつ，“ sconst セクションのデータ配置 ” の “ 配置する [-Xsconst] ” にチェックしない場合（デフォルト），型修飾子 const を指定したデータ，および文字列リテラルは，すべて .const セクションへ割り付けられます。

(3) レジスタモード

レジスタ・モードを設定します。設定できるのは，次のとおりで，デフォルトは 32 レジスタ・モードです。

- 22 レジスタ・モード
- 26 レジスタ・モード
- 32 レジスタ・モード

レジスタ・モードの設定は，リンカにも認識されるため，ライブラリのリンク時に適切なレジスタ・モードのライブラリを参照します。

レジスタ・モードについての詳細は“ CA850 ユーザーズ・マニュアル C 言語編 ”を参照してください。

(4) プロローグエピローグランタイム

関数のプロローグ／エピローグ処理をランタイム・ライブラリ呼び出しによる処理にするかを設定します。“ 使用 ” を指定した場合，関数のプロローグ／エピローグ処理がランタイム・ライブラリ呼び出しになります。

なお,“ デフォルト ”が選択されているときは，[最適化とデバッグ情報] タブのオプションで，最適化タイプとして “ より高度な最適化（実行速度優先） ” を指定している場合 “ 未使用 ” となり，その他のタイプの場合 “ 使用 ” となります。

この設定はソース・ファイルごとに設定することができます。“ デフォルト ” が選択されているときは，全体にかかる設定と同じ設定になります。

関数のプロローグ／エピローグ・ランタイム・ライブラリについての詳細は “ CA850 ユーザーズ・マニュアル C 言語編 ” を参照してください。

(5) switch の出力コード

switch 文のコード出力方式を指定します。

- デフォルト
  コンパイラが最適と思われる形式を自動判断します。
- if-else
  case 文の並びに沿った if-else 文と同じ形で出力します。頻度が多い順に case 文を書いているときやラベル数が少ないときは，これを選択します。上から順に比較するので，頻繁に合致する case 文を先に記述すると余計な比較が減り，実行速度向上につながります。
- バイナリサーチ
  バイナリ・サーチ形式で出力します。バイナリ・サーチ・アルゴリズムに用いて合致する case 文を探します。ラベル数が多いときにこれを選択すると，どの case 文も同じくらいの速さで見つけることができます。
- テーブル分岐
  テーブル・ジャンプ方式で出力します。case 文の値を基にインデックス化したテーブルを参照し，switch 文の値により case ラベルを選択し，処理を行います。どの case 文にも同じくらい速く分岐します。
  ただし，case 値が連続していないときは無駄な領域ができます。

(6) 構造体パッキング

-Xpack=*num* オプションで指定する構造体パッキングを設定します。このオプションを用いることにより，構造体メンバをメンバの型に応じてアライメントすることなく，指定したアライメントを用いることができます。データ・サイズは小さくすることができますが，コード・サイズは大きくなります。指定できるの値は“ 1 バイト ”,“ 2 バイト ”,“ 4 バイト ”,“ 8 バイト ” です。デフォルトは“ 8 バイト ” です。[注]

注　本バージョンでは“ 4 バイト ” と“ 8 バイト ” を指定したときの動作は同じになります。

C 言語ソース・ファイル中に #pragma 指令で構造体パッキング指定がある場合に，このオプションを指定した場合，最初の #pragma 指令が出現するまでは，オプション指定値がすべての構造体に適用されます。それ以降は，#pragma 指令の値が適用されます。

ただし，#pragma 指令の出現後でも指定がデフォルトになった部分は，オプション指定値が適用されます。このオプションは，-Xi オプションと同時指定はできません。

また，このオプションには，次の注意点があります。

- C 言語ソース・ファイル中に #pragma 指令で構造体パッキング指定がある時に -Xpack オプションを指定した場合，最初の #pragma pack 指令が出現するまではオプション指定値がすべての構造体に適用されます。それ以降は #pragma 指令の値が適用されます。
  ただし，#pragma 指令の出現後でも指定がデフォルトになった部分は，オプション指定値が適用されます。

また，V850 ／ V850Ex ／ V850E2 コア製品ミス・アライン・アクセス禁止の設定の CPU をご使用の場合，次の制限があります。この制限は，#pragma pack についても同様です。

- 構造体メンバのアドレスを取得すると正しく取得できません。
- ビットフィールドへのアクセスは，そのメンバの型で読み込むため，データ領域もアクセスします。
  ビットフィールドの幅が，メンバの型以下の場合，メンバの型で読み込むのでオブジェクトの外部にアク

セスします。実行上通常は問題がありませんが，I/O などがマップされていた場合，不正なアクセスとなる場合があります。

構造体パッキングについての詳細は“ CA850 ユーザーズ・マニュアル C 言語編 ”を参照してください。

(7) ポインタのバイトアクセス [-Xbyte]

構造体の間接アドレス・アクセスをバイト単位でアクセスします。構造体パッキング機能で，制限にかかるような場合に使用します。

(8) ソースのコメントの出力 [-Xc]

出力するアセンブリ言語ソース・ファイル中に C 言語ソース・プログラムをコメントとして出力します。“ アセンブルリストの出力 ”指定時にのみ指定可能です。

ただし，出力されるコメントは，あくまで参考であり，厳密にはコードと対応していない場合もあります。たとえば，グローバル変数とローカル変数，関数宣言などのコメントの出力位置がずれることがあります。また，最適化によりコードが削除され，コメントのみが残ることもあります。

(9) strcpy の展開 [-Xi]

配列（文字列を含む），および構造体の整列条件を 4 バイトとし，関数 strcpy( )，または strcmp( ) の呼び出しをブロック転送に変換します。オブジェクトの実行速度は高速になりますが，コード・サイズは増大します。このオプションは，strcpy( )，または strcmp( ) の第二引数が文字列の場合にのみ変換します。また，第一引数は，プログラム側で 4 バイトに整列されている必要があります（第二引数は文字列のため，ca850 が整列しています）。

このオプションは，-Xpack オプションと同時指定はできません。

(10) 割り込みの分岐命令の変更 [-Xj]

C 言語で定義された通常の割り込み関数に対し，jmp 命令を用います。関数本体が，jr 命令では分岐できない範囲（± 1M バイト以上）にあり，ld850 がエラーを出力する場合，このオプションを用いてコンパイルし直します。このオプションを省略した場合，jr 命令を用います。

Flash ／外付け ROM 再リンク機能でブート側からフラッシュ側の関数を呼び出す場合には，このオプションは指定できません。詳細は「5.6 フラッシュ／外付け ROM 再リンク機能」を参照してください。

(11) マスクレジスタの使用 [-Xmask_reg]

マスク・レジスタ機能を使用します。この機能を使用すると，ca850 は r20 に 8 ビットのマスク値 0xff，r21 に 16 ビットのマスク値 0xffff が設定されているものとしてコードを出力します。マスク・レジスタ（r20，r21）へのマスク値の設定は，スタート・アップ・ルーチン等，ユーザ・プログラムで行う必要があります。

V850 マイクロコントローラでは，バイト・データ，ハーフワード・データをメモリからレジスタにロードする場合，最上位ビットの値によりワード長へ符号拡張します。このため，unsigned char，unsigned short 型データの演算では，上位ビットのマスク・コードが生成される場合があります。また，演算結果をレジスタ変数へストアする場合，符号なしバイト・データ，符号なしハーフワード・データでは，上位ビットをクリアするためにマスク・コードが生成されます。どちらの場合も，ワード・データに切り替えれば回避できますが，ワード・データにできず，マスク・コードが生成される場合，マスク・レジスタ機能を用いることにより，コード・サイズの削減ができます。

ただし，マスク・レジスタ機能を利用するかどうかの判断には，利用する側で次の点を十分考慮する必要があります。

- マスク・コードが多く出力されるプログラムであるか。
- マスク・レジスタとして使用するため，レジスタ変数用レジスタが 2 本少なくなるが，その影響はないか。

なお，このオプションを指定したとき，マスク・レジスタを使用したオブジェクトとしないオブジェクトが混在する場合，ld850 でエラーとなります。また，32 レジスタ・モードのとき，ld850 に -mask_reg が渡されます。これにより，リンカによる標準ライブラリの検索は，標準フォルダより先に，マスク・レジスタ用フォルダで行います。

マスク・レジスタについての詳細は “ CA850 ユーザーズ・マニュアル C 言語編 ” を参照してください。

なお，このオプションは，ソース個別の指定では選択できません。常に全体のオプションとして設定します。

注 C 言語ソース・ファイルとアセンブリ言語ソース・ファイルの両方を使用するアプリケーション，またはアセンブリ言語ソース・ファイルのみを使用するアプリケーションの場合であっても，“ マスクレジスタの使用 ” は，コンパイラ・オプション，アセンブラ・オプションの両者で指定する必要があります。

(12) word の bit 命令変更の抑止 [-Xno_wordbitop]

ld.w ／ ld.h，st.w ／ st.h 命令を 1 ビット操作命令（set1，clr1，tst1，not1）へ置き換える動作を禁止します。デバッグ時に変数の read ／ write イベントを設定する場合，1 ビット操作命令に置き換わっていると，イベントが発生しないことがあります。この場合，このオプションを指定すると，ld.w ／ ld.h，st.w ／ st.h 命令のままとなり，デバッグがしやすくなります。

(13) word switch テーブル [-Xword_switch]

switch 文中の case ラベルに対する分岐テーブルを，1 ラベルあたり 4 バイトで生成します。長い switch 文のためコンパイル・エラーとなる場合，このオプションを指定します。このオプションを指定しない場合，分岐テーブルを 2 バイトで生成します。

(14) コマンドラインオプション

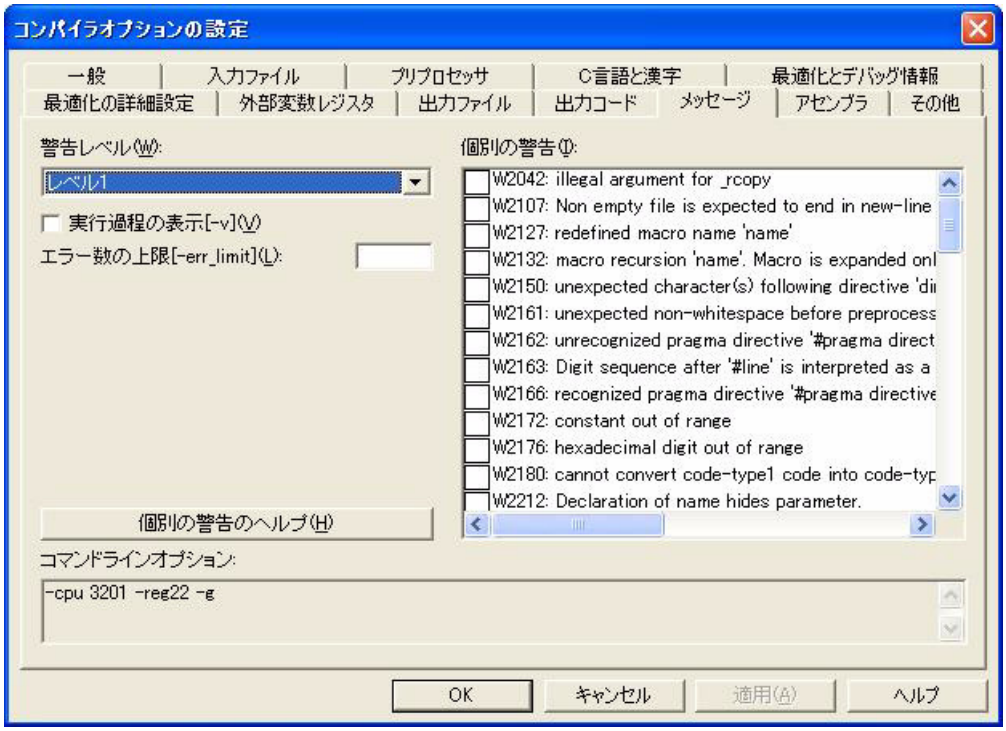
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[メッセージ]

メッセージに関するオプションの設定を行います。

図 3 － 21 [コンパイラオプションの設定] ダイアログ：[メッセージ] タブ



(1) 警告レベル

出力する警告メッセージのレベルを指定します。

出力しない [-w]	警告メッセージを抑止する
レベル 1	通常の警告メッセージを出力する（デフォルト）
レベル 2[-w2]	詳細な警告メッセージを出力する

(2) 実行過程の表示 [-v]

ビルド時に ca850 の実行状況の詳細を，アウトプット・ウインドウに表示します。チェックすると，コンパイラ内部の各フェーズの実行状況まで表示します。

(3) エラー数の上限 [-err_limit]

エラー・メッセージの最大出力数を指定します。15 から 50 までの 10 進数を指定します。このオプションを省略した場合，15 とします。

(4) 個別の警告

このリスト・ビューは，個別の警告メッセージの表示を制御するものです。各警告メッセージの左にあるアイコンが“ ON ”の場合は，その警告を表示（-won），“ OFF ”の場合は非表示（-woff）になります。

アイコンはダブル・クリック，またはスペース・キーで“ ON ”→“ OFF ”→“ ”の順に切り替わります。このオプションはソース・オプションの設定では，設定できません（全体オプションが常に設定されます）。

次のオプションは，プロジェクトの読み込み時にこれらのメッセージのオプションに変換されます。

-wbitfield_align	W2306
-wbitfield_type	W2302
-wcallnodecl	W2215
-Xd	W2231
-wnopic	W2231
-wpragma	W2162
-wsharp	W2161

指定可能なメッセージを次に示します。

表 3 － 7 指定可能なメッセージのメッセージ番号

W2042，W2017，W2127，W2132，W2150，W2161，W2162，W2163，W2166，W2172，W2176，W2180，W2212，W2215，W2216，W2222，W2231，W2244，W2254，W2267，W2287，W2289，W2291，W2293，W2302，W2306，W2373，W2380，W2416，W2520，W2521，W2525，W2527，W2554，W2555，W2606，W2607，W2621，W2634，W2635，W2637，W2643，W2656，W2671，W2683，W2684，W2690，W2691，W2699，W2700，W2703，W2704，W2710，W2711，W2730，W2731，W2740，W2741，W2742，W2743，W2744，W2748，W2761，W2782

なお，[個別の警告のヘルプ] ボタンで警告メッセージのオンライン・ヘルプを参照することができます。

(5) コマンドラインオプション

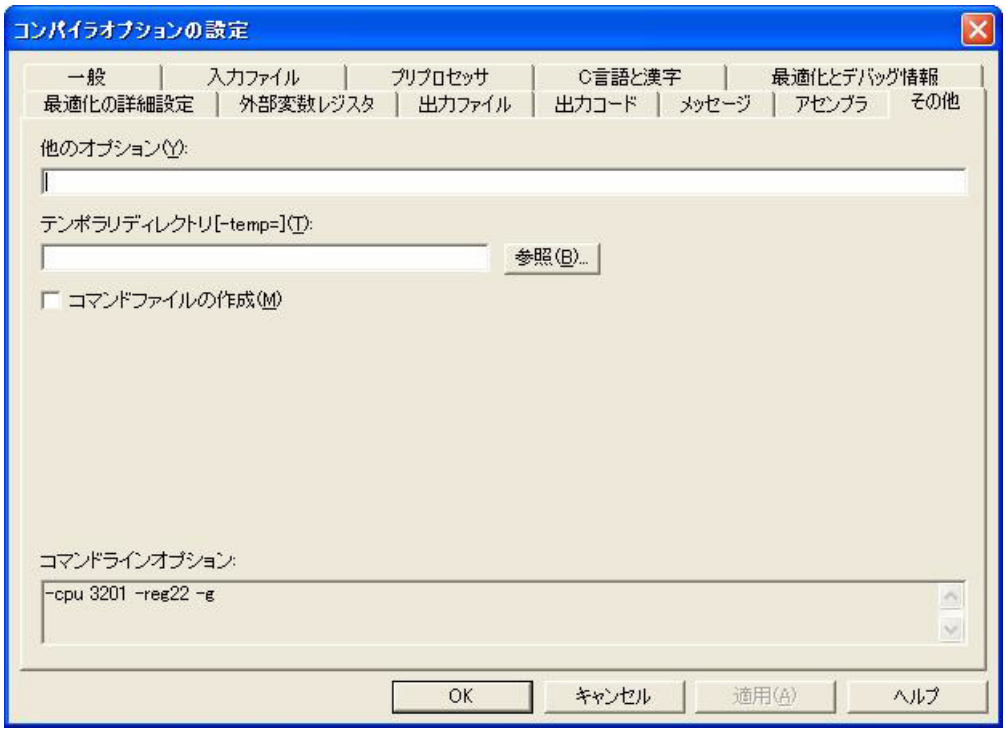
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[その他]

その他のオプション設定を行います。

図 3 － 22 [コンパイラオプションの設定] ダイアログ：[その他] タブ



(1) 他のオプション

前記の “ コンパイラオプションの設定 ” では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

現在のところ，“ 他のオプション ” で指定できるオプションは次のとおりです。

- -Xv850patch

これ以外のオプションを指定することもできますが，現状未サポートです。

(2) テンポラリディレクトリ

内部的に用いるテンポラリ・ファイルを生成する作業用フォルダを指定します。このオプションを省略した場合，環境変数 TEMP，またはカレント・ドライブのルートに生成します。ハード・ディスクの容量不足などで，テンポラリ・ファイルを生成することができないためのエラーが発生した場合，このオプションで回避できます。指定したフォルダが存在しない場合は，フォルダを作成するかどうか確認するメッセージ表示されます。

(3) コマンドファイルの作成

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。チェックされている場合は，オプション文字列はコマンド・ファイル（「3.7.2 コマンド・ファイル」参照）に出力されるため，文字列の制限を意識する必要がなくなります。数多くのオプションを設定し，オプションを認識しきれない場合などに，チェックしてください。

なお，このオプションはデフォルトではチェックされていません。

(4) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[備考]

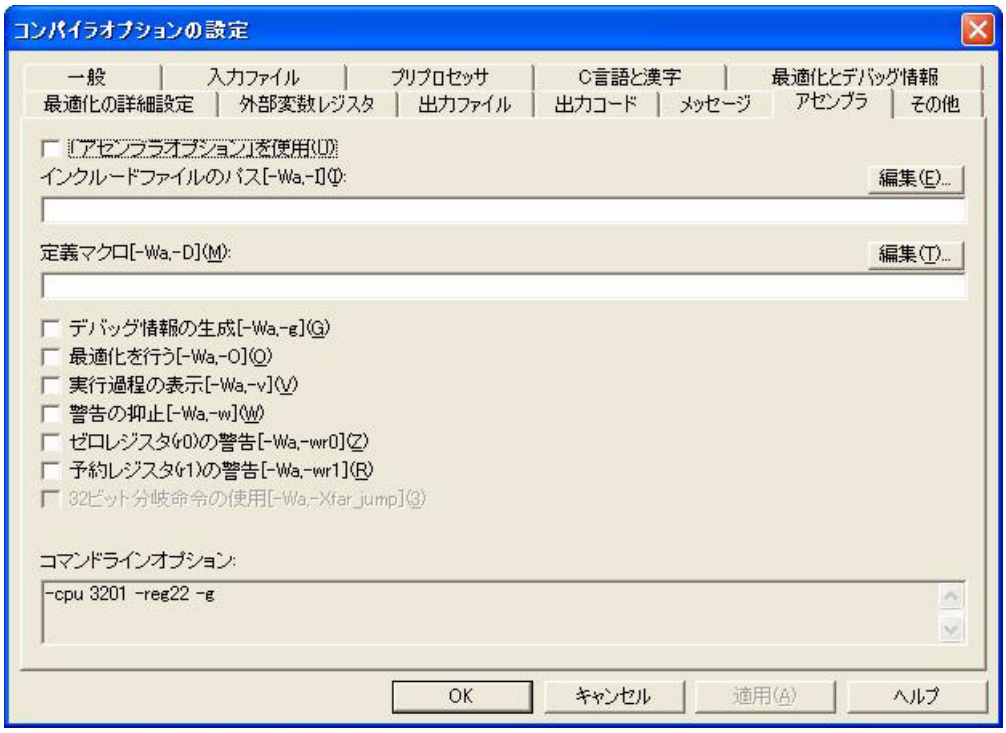
(a) [ソースオプションの削除] ボタン

ソース・ファイル個別のオプション設定を行っている場合に選択可能なボタンです。プロジェクト全体のオプション指定時には表示されず，選択できません。個別オプションを無効にし，全体のオプション設定を有効にしたい場合は，このボタンをクリックします。

[アセンブラ]

C 言語ソース・ファイルのアセンブル時に使用するアセンブラのオプションの設定を行います。

図 3 － 23 [コンパイラオプションの設定] ダイアログ：[アセンブラ] タブ



(1) 「アセンブラオプション」を使用

C 言語ソース・ファイルのアセンブル時に，このページのアセンブラ・オプションではなく [アセンブラオプションの設定] ダイアログのオプションを使用するかどうかを設定します。チェックされている場合は，[アセンブラオプションの設定] ダイアログのオプションを使用します。

(2) インクルードファイルのパス [-Wa,-I]

インクルードファイルのパスを設定します。複数のパスを指定する時は，パスを“;（セミコロン）”で区切ります。このオプションはソース個別では設定できません（全体オプションが常に設定されます）。

なお，このオプションに設定したパス以外に，システムで設定したオプションがコマンド・ライン・オプションに設定される場合があります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，パス項目をダイアログ上で編集することができます。

(3) 定義マクロ [-Wa,-D]

定義マクロを設定します。複数のマクロを指定する場合は，マクロを“;（セミコロン）”で区切ります。

[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，定義マクロ項目をダイアログ上で編集することができます。

(4) デバッグ情報の生成 [-Wa,-g]

デバッグ情報を生成するかどうかを設定します。チェックされている場合は，-Wa,-g オプションを指定したのと同じになります。

(5) 最適化を行う [-Wa,-O]

最適化を行うかどうかを指定します。チェックされている場合は，-Wa,-O オプションを指定したのと同じになります。

(6) 実行過程の表示 [-Wa,-v]

実行過程の表示するかどうかを設定します。チェックされている場合は，-Wa,-v オプションを指定したのと同じになります。

(7) 警告の抑止 [-Wa,-w]

警告メッセージを抑止するかどうかを指定します。チェックされている場合は，-Wa,-w オプションを指定したのと同じになります。

(8) ゼロレジスタ（r0）の警告 [-Wa,-wr0]

r0 レジスタをデスティネーション・レジスタに使用した場合の警告メッセージを抑止するかどうかを指定します。チェックされている場合は，-Wa,-wr0+ オプションを指定したのと同じになります。また，グレーの場合は，-Wa,-wr0- オプションを指定したのと同じになります。

(9) 予約レジスタ（r1）の警告 [-Wa,-wr1]

r1 レジスタを使用した場合の警告メッセージを抑止するかどうかを指定します。チェックされている場合は，-Wa,-wr1+ オプションを指定したのと同じになります。また，グレーの場合は，-Wa,-wr1- オプションを指定したのと同じになります。

(10) コマンドラインオプション

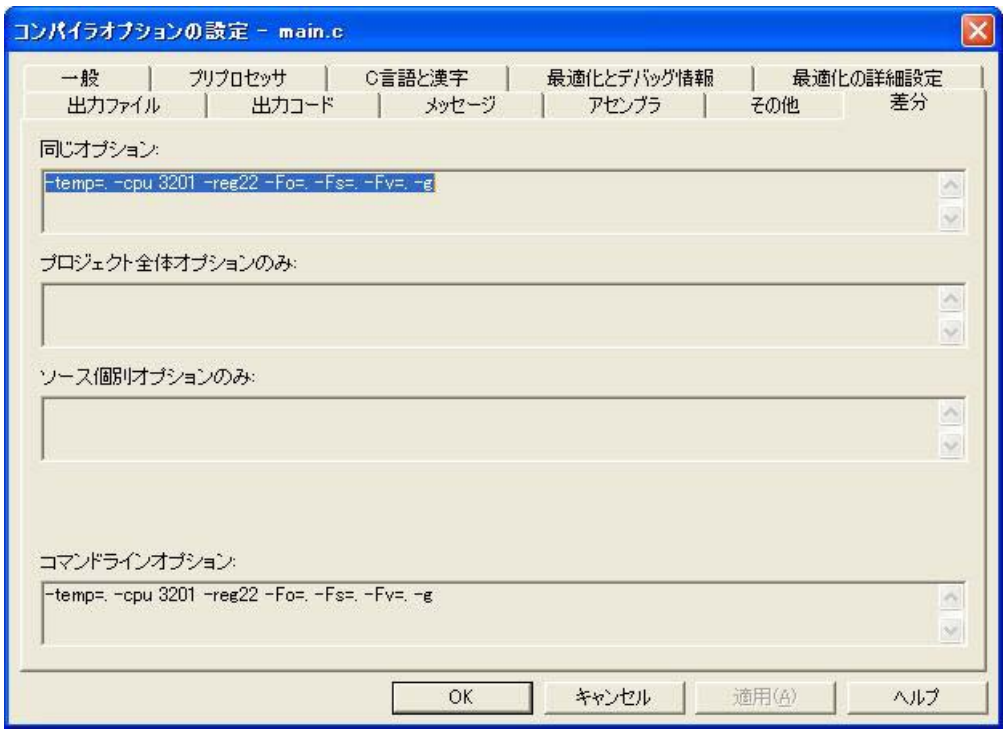
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[差分]

プロジェクト全体オプションとソース個別オプションの差分を表示します。

図 3 － 24 [コンパイラオプションの設定] ダイアログ：[差分] タブ



(1) 同じオプション

プロジェクト全体オプションとソース個別オプションの双方で指定されているオプションがコマンド・ライン形式で表示されます。

なお，このエリアは参照用のため，書き込みはできません。

(2) プロジェクト全体オプションのみ

プロジェクト全体オプションで指定されていてソース個別オプションでは指定されていないオプションがコマンド・ライン形式で表示されます。

なお，このエリアは参照用のため，書き込みはできません。

(3) ソース個別オプションのみ

ソース個別オプションで指定されていてプロジェクト全体オプションでは指定されていないオプションがコマンドライン形式で表示されます。

なお，このエリアは参照用のため，書き込みはできません。

(4) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[備考]

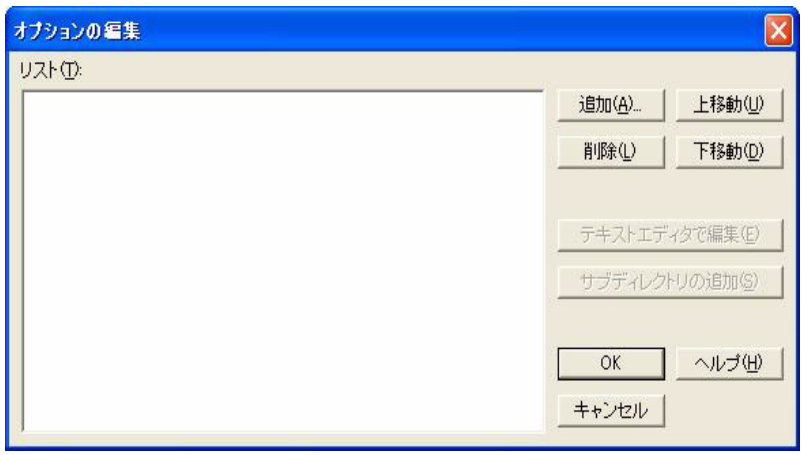
(a) [ソースオプションの削除] ボタン

ソース・ファイル個別のオプション設定を行っている場合に選択可能なボタンです。プロジェクト全体のオプション指定時には表示されず，選択できません。個別オプションを無効にし，全体のオプション設定を有効にしたい場合は，このボタンをクリックします。

3.6.3 [オプションの編集] ダイアログ

[オプションの編集] ダイアログは，項目をリストで編集します。

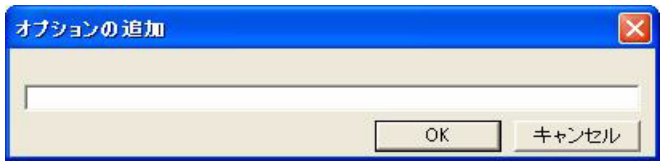
図 3 － 25 [オプションの編集] ダイアログ



(1) [追加] ボタン

リストの項目を追加します。ファイルやフォルダを指定する項目の場合は，それぞれの参照用のダイアログがオープンします。それ以外の場合は，内容を入力する次の [オプションの追加] ダイアログがオープンします。

図 3 － 26 [オプションの追加] ダイアログ



(2) [削除] ボタン

選択中のリストの項目を削除します。

(3) [上移動] ボタン

選択中のリストの項目を上に移動します。

(4) [下移動] ボタン

選択中のリストの項目を下に移動します。

(5) [テキストエディタで編集] ボタン

次の場合は，テキスト・エディタで編集することができます。

- [コンパイラオプションの設定] ダイアログの [入力ファイル] タブの “ セクションファイル [-Xsec_file] ”
- [コンパイラオプションの設定] ダイアログの [入力ファイル] タブの “ Far Jump ファイル [-Xfar_jump] ”

(6) [サブディレクトリの追加] ボタン

次の場合は，選択中のリストの項目にサブフォルダを追加できます。

- [コンパイラオプションの設定] ダイアログの [プリプロセッサ] タブの“ インクルードファイルのパス [-I] ”
- [コンパイラオプションの設定] ダイアログの [アセンブラ] タブの“ インクルードファイルのパス [-Wa,-I] ”
- [アセンブラオプションの設定] ダイアログの [オプション] タブの “ インクルードファイルのパス [-I] ”
- [リンカオプションの設定] ダイアログの [ライブラリ] タブの “ ライブラリのパス [-L] ”

3.7 注意事項

3.7.1 オプションの複数指定

これらのオプションの中には，他のオプションと同時に指定された場合，無効になるものがあります。次に示した“ > ”の右側に置かれたオプションは，左側に置かれたオプションと同時に指定された場合，無効になります。

- -E > -P
- -U > -D
- -E ／ -P > -G ／ -L ／ -O ／ -R ／ -S ／ -Wc ／ -a ／ -c ／ -I ／ -m ／ -o

  前処理で終了するため，フロントエンド以降のモジュールに関するオプションは無効になります。
- -S > -L ／ -R ／ -W[a|l] ／ -a ／ -c ／ -I

  コード生成部，または機種依存最適化部で終了するため，as850 以降のモジュールに関するオプションは無効になります。
- -V ／ -help

  あとから指定された方が無効になります。また，このオプションを指定した場合，他のオプションはすべて無効になります。
- -c > -L ／ -R ／ -Wl ／ -l

  as850 で終了するため，ld850 以降のモジュールに関するオプションは無効になります。
- -m > -G ／ -L ／ -O ／ -R ／ -S ／ -Wc ／ -a ／ -c ／ -I

  フロントエンドで終了するため，プリオプティマイザ以降のモジュールに関するオプションは無効になります。
- -Og ／ -O ／ -Os ／ -Ot > -a ／ -Fv

  -Og ／ -O ／ -Os ／ -Ot が指定された場合，正しく表示しない場合があります。
- -Od ／ -Ob ／ -Og ／ -O ／ -Os ／ -Ot

  あとから指定されたものが有効となります。
- -w ／ -w[1|2]

  先に指定された方が無効となります。

3.7.2 コマンド・ファイル

コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなく，ファイルに記述して指定するものです。ca850 は，コマンド・ファイルの内容をコマンド・ラインの引数のように扱います。また，コマンド・ファイルでは，指定する引数を複数行に分けて記述できます。ただし，オプションやファイル名などが 2 行に分かれないようにしてください。また，コマンド・ファイルのネストはできません。

コマンド・ファイルは，次の文字については特殊文字として扱います。

"（ダブルクォーテーション）	次の "（ダブルクォーテーション）までの文字列を連続した文字列として扱います。
#（シャープ）	行頭に指定した場合は行末までをコメントとして扱います。
^（ハット）	直後の文字を特殊文字として扱いません。

なお,特殊文字自体はコマンドファイルを指定したca850のコマンド・ラインの中に含まれずに削除されます。

備考 as850，ar850，hx850，dump850，dis850，および romp850 は，"（ダブルクォーテーション）のみ使用できます。

コマンド・ファイルの例

```
-Dtest        … #define test を記述
-o object     … オブジェクト・ファイル名を指定
a.c           … コンパイルするファイルを指定
```

コマンド・ファイルの指定例

```
> type cfile
       -cpu 3201 -c -Os file.c ← コマンド・ファイルの内容
> ca850 @cfile ← ca850 -cpu 3201 -c -Os file.c と同じ動作をする
```

3.7.3 効率的な最適化の仕方

ここでは，-O オプションで指定可能な最適化処理の主な内容と，効率的に最適化を行うための指定を示します。

図 3 － 27 最適化処理と項目

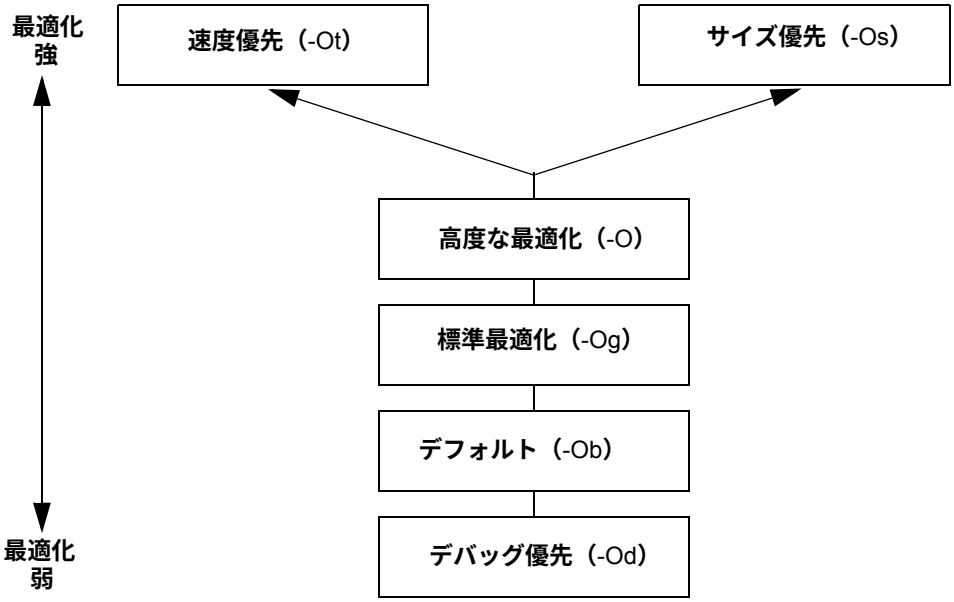


表 3 － 8 最適化処理と項目

オプション	効果			
最適化機能	デバッグ	コード効率	実行速度	コンパイル時間
-Od：デバッグ優先	レベル 4	レベル 1	レベル 1	レベル 3
-Ob：デフォルト	レベル 3	レベル 2	レベル 2	レベル 3
-Og：標準最適化	レベル 3	レベル 3	レベル 3	レベル 3
-O：高度な最適化	レベル 2	レベル 4	レベル 4	レベル 2
-Os：より高度な最適化（サイズ優先）	レベル 1	レベル 5	レベル 4	レベル 2
-Ot：より高度な最適化（実行速度優先）	レベル 1	レベル 4	レベル 5	レベル 1

表中の表現は次のとおりです。

デバッグ	最適化が強くなると，C 言語ソース行の削除や同一処理行を一箇所にまとめる最適化を行うことから，ブレークポイントが設定できる箇所が少なくなる傾向があります。変数もメモリからレジスタを割り当てる確率も向上します。 最適化によるブレークポイントの現象や変数のレジスタ割り当てが少ないものを “ レベル 4 ” とし，最も傾向が強いものを “ レベル 1 ” としています。“ レベル 1 ” となっているものであっても，デバッグは可能です。
コード効率	ROM サイズ効率について，“ レベル 1 ～レベル 5 ” で分類しています。 最も ROM サイズが小さくなるオプションは，-Os ですが，コンパイル時間がかかります。ROM 容量に比較的余裕のある場合は，-Og，または -O オプションを指定してください。
実行速度	実行速度について，“ レベル 1 ～レベル 5 ” で分類しています。 モジュール全体として ROM 容量を小さくしたいが，クリティカルな関数のみ実行速度をさらに向上したい場合は，ファイル単位で -Ot オプションを指定してください。
コンパイル時間	コンパイル時間について，“ レベル 1 ～レベル 3 ” で分類しています。 -O，-Os，-Ot は強力な最適化を行うことから，コンパイル時間はこれら以外のオプションに比べて遅くなります。

(1) -Od：デバッグ優先

基本ブロック[注]内最適化を行います。基本ブロック内最適化とは，基本ブロック内で掌握できる情報を用いて行う最適化です。

- 定数計算や式の変形，
- 基本ブロック内の共通部分式の認識，
- 基本ブロック内の複写の伝播

などがあります。

なお，この基本ブロック内最適化は，コンパイル時にデフォルトで行われます。たとえば，“ 定数のみの演算式をコンパイル時に結果の定数に置き換える ” という最適化を指します。

CA850 では最も弱い最適化となります。この最適化は，CA850 Ver.2.4x のデフォルトの最適化と同等の最適化レベルとなります。

注 必ず先頭の命令から入り，最後の命令以外からは分岐しない最大の命令の並びです。

(2) -Ob：デフォルト

基本ブロック内最適化と自動変数のカラリング・レジスタ割り付けを行います。

- 自動変数をレジスタとして割り付けます。
  デバッグ時の影響はありません。

CA850 Ver.2.50 以降のデフォルトです。レジスタ割り付けが高機能となるため，-Od よりも無駄なコードが削除されます。

(3) -Og：標準最適化

基本ブロック内最適化，およびカラリング・レジスタ割り付けに加え，関数内で掌握できる情報を用いて，次の最適化を行います（代表的な物のみ記載）。

- 共通な演算を見つけ，まとめて処理する命令列を出力する。
- ループ内の値が変化しない代入文をループ外へ移動する。
  ステップ実行や，ブレークポイントがユーザの意図どおり設定できないことがあります。
- 冗長な代入文の削除。
  削除された行のブレークポイントは設定できません。

(a) 外部変数をレジスタ割り付けする。
  デバッグ時にメモリへの read ／ write ブレークが正確に行えないことがあります。

(b) ca850 により命令を並び替えてレジスタ／フラグのハザードを回避する最適化を行います。
  デバッグ時の影響はありません。

高度な最適化を行う場合に比べ，コンパイル速度も速く，コード効率／実行速度は，CA850 の最適化で中間の性能となります。ROM 容量に比較的余裕のある場合は，このオプションの設定を推奨します。

(4) -O：高度な最適化

-Og までの最適化に加え，次の最適化を行います（代表的な物のみ記載）。

- 関数の引数の参照がないことを判断し，引数への値の代入コードを削除します。
  デバッグ時の影響はありません。
- 実行回数が 1 回のループのみは展開し，終了条件判断のオーバーヘッドを回避します。
  デバッグ時の影響はありません。
- ラベルの整列や関数の先頭での 4 バイト整列を抑制します。
  デバッグ時の影響はありません。
- 未参照ラベルの削除を行います。
  削除対象のラベルにブレークポイントが設定できなくなります。
- 不要命令の削除を行います。
  ブレークポイントやステップ実行がユーザの意図どおり設定できないことがあります。
- ピープホール最適化（効率の良い命令列への 5 命令以内の並び替え）を行います。
  ブレークポイントやステップ実行がユーザの意図どおり設定できないことがあります。

この最適化は，CA850 Ver.2.4x のオブジェクト・サイズ優先 -Os に相当します。

なお，このオプションは，CA850 Ver.2.4x で行われていた一度しか参照されない静的関数のインライン展開は行われません。

(5) -Os：より高度な最適化（サイズ優先）

-O の処理を最適化ができなくなるところまで，最適化を行います。CA850 がサポートしている最適化のうち，コード・サイズを増加させない最適化のすべてを行い，できるかぎりサイズを小さくする，最強のオブジェクト・サイズ優先最適化オプションです。

ただし，アプリケーションの内容によっては，-Os に加え，次のオプションや機能を使用することにより，最適化をさらに強化できる場合があります。

なお，アプリケーションの内容によっては，上記オプションに加えて，次のオプションや機能を使用することにより，最適化を強化できる場合があります。

- -Wi,-O4 を指定する

  データ・フロー解析を行い，最適化を強化します。ただし，コンパイル時間はかなり増大する可能性があります。
- マスク・レジスタを使用する

  unsigned char，unsigned short 型の演算のためにマスク・コードが多発するアプリケーションの場合，マスク・レジスタ機能を利用すると，コード・サイズを削減できます。

  ただし，マスク・レジスタ機能では，使用できるレジスタ変数用レジスタが 2 本少なくなります。
- セクション・ファイルを使用する

  データは，内蔵メモリや，gp ／ r0 相対で 1 命令参照するセクションに割り当てると，コード・サイズが削減され，高速化できます。プログラムでデータのセクション割り当てを行っていない場合，コンパイル時にセクション・ファイル（「9.1 セクション・ファイル」参照）で，[tidata.byte] ／ [tidata.word] ／ [sidata] ／ [sedata] ／ [sconst] ／ [sdata] に割り当てます。

ca850 でコード・サイズに着目した最適化では最もサイズが小さくなります。CA850 Ver.2.4x のオブジェクト・サイズ優先 -Os ＋オプショナル最適化 -Ol に相当します。

なお，このオプションは，CA850 Ver.2.4x で行われていた一度しか参照されない静的関数のインライン展開は行われません。

(6) -Ot：より高度な最適化（実行速度優先）

実行速度優先で最適化を行います。データ処理アプリケーションなど，サイズを犠牲にしてでも実行時間を短縮したい場合に使用します。

-O までの最適化に加えて，

- ラベルの 4 バイト整列
- 関数の先頭での 4 バイト整列

の抑制の最適化を行い，さらに，

- テール・リカージョン最適化
- インライン展開
- ループ展開

を行います。

テール・リカージョン最適化では，関数の終わりの return 文が，その関数自身の呼び出しである場合，その関数をループに変換し，関数呼び出しによるスタック量を減らします。

インライン展開では，関数呼び出しの部分に関数本体を展開し，最適化の可能性を高め，呼び出しによるオー

バーヘッドを防ぎます。

ループ展開では，ループ本体を複数回展開し，最適化の可能性を高め，条件判断や分岐によるオーバーヘッドを防ぎます。

インライン展開，ループ展開では，オブジェクト・サイズは大きくなりますが，実行速度の向上が期待できます。

なお，-Ot を指定し，ラベル定義している asm 文を含む関数を使用した場合，関数の定義部分とインライン展開された部分とで，同じラベルが定義されることになります。この場合，ラベルの多重定義エラーになるため注意が必要です。また，#pragma block_interrupt，#pragma interrupt，#pragma rtos_task，#pragma text 指定された関数は，インライン展開しません。その際，メッセージも出力しません。

スタック・フレームの操作のような，インライン展開されることを予期していない asm 文を含む関数を使用した場合には，不正な関数フレーム操作などが起こるために，実行エラーとなることがあります。

インライン展開についての詳細は “ CA850 ユーザーズ・マニュアル C 言語編 ” を参照してください。

注意 より高度な最適化（実行速度優先）によってサイズが大きくなりすぎる場合，“ -Wp,-Gnum ” や “ -Wo,-Ol[num] ” オプションで，インライン展開やループ展開を調整してください。オプションと関係なく特定の関数のみインライン展開するには，#pragma inline を使用します。これにより “ サイズ優先 ” を指定しつつ，特定の関数呼び出しのみ実行速度を優先できます。

なお，アプリケーションの内容によっては，-Os 指定のときと同様にマスク・レジスタなどで実行速度優先の最適化を強化できる場合があります。

また，次の機能によっても，強化できる場合があります。

- strcpy() を展開する

  文字列コピー関数の strcpy() を多用するアプリケーションの場合，“ strcpy の展開 ” を行う -Xi オプションを指定すると，実行時間が短縮されます。ただし，サイズは増大します。
- -Wp,-r を指定する

  ソース・ファイルをマージしたインライン展開の結果，不要な関数が生じる場合があります。そのような場合，“ -Wp,-r ” オプションを指定すると，不要な関数が削除され，サイズが削減される場合があります。

CA850 で実行速度に着目した最適化では最も実行速度が短くなります。CA850 Ver.2.4x の実行速度優先 -Ot ＋オプショナル最適化 -Ol に相当します。

以上のように，CA850 の最適化には，いくつかのレベル／項目がありますが，最適化を指定する場合，基準となるのは，次の選択です。

- サイズを重視する
- サイズを犠牲にしても実行速度を重視する

ほとんどの最適化機能は，サイズ削減と同時に実行速度を向上させますが，一部の機能を使用するかどうかで，サイズをより重視するか，実行速度をより重視するかが決まります。

3.7.4 最適化によるデバッグへの影響

最適化を行うと，ソース・デバッガを使用する際，次のような影響があるので注意してください。

(1) 最適化による式の変形（複写の伝播や共通部分式の認識）によって，ソース・プログラム中での出現箇所で“変数参照”が起こらなくなり，変数の read ／ write イベントがユーザの意図どおり発生しない場合があります。

(2) 文の共通化や削除，並び替えが行われると，ステップ実行やブレークポイントがユーザの意図どおり設定できないことがあります。

(3) 変数の生存範囲（プログラム中でその変数を参照可能な範囲），変数の位置（レジスタやメモリ上の位置）が変更される可能性があります。

(4) 文の削除が起こった場合，その文にはブレークポイントを設定できません。

(5) 文の移動や分割，統合により，実行命令の順序が入れ替わった[注]場合，入れ替わった行とそれらの行の間にある行は，1 つのまとまりとして扱われ，途中のブレークポイントの設定やステップ実行ができなくなる場合があります。

注 あるソース行に対する実行命令のアドレスが，それ以前の行に対する実行命令のアドレスより小さくなった，あるいは，それ以後の行に対する実行命令のアドレスより大きくなったことをいいます。

(6) if-else の各場合で実行命令の順序が入れ替わったり，ループ展開で実行命令の順序が入れ替わった場合，(2) と同様にステップ実行などができなくなる場合があります。

(7) 自動変数はすべて有効範囲（スコープ）が関数全体とみなされます。しかし，その変数がレジスタへ割り付けられた場合，スコープ内であっても最適化により削除され，見えなくなる可能性があります。これは，スコープ内でその変数が“局所的に”使用されている場合や，最適化の結果として局所化された場合に起こります。

例

```
void f(void)
{
        int     a;              /* 関数内で有効 */
                :
        /* アドレス 1 */
                :               /* a はアドレス 1 ～アドレス 2 の範囲でのみ使用 */
        /* アドレス 2 */
                :
}
```

この例では，a のスコープは関数 f() 内全体です。しかし，a はアドレス 1 ～アドレス 2 間に限定して使用されています。このとき，a がレジスタに割り付けられ，最適化によりスタック・フレームから削除されると，a はアドレス 1 ～アドレス 2 の区間外では見えなくなります。この現象は，レジスタを効率的に使用するために，a の見える区間外では，a が割り付けられたレジスタに他の変数を割り付ける結果として生じます。

(8) コンパイル時，デバッグ情報の処理でメモリを大量に消費するため，“out of memory”となる可能性があります。

(9) インライン展開が行われた部分は 1 つのまとまりとして扱われ，ステップ実行はできません。

(10) **ループ展開が行われた部分はループ本体部分が 1 つのまとまりとして扱われ，その内部をステップ実行はできません。また，本体部分のまとまりで停止する回数は，展開前のループ数ではなく，展開後のループ数となります。**

(11) **外部変数にレジスタを割り付けた場合は指定した外部変数のデバッグ情報が削除されるため，最適化デバッグはできなくなります。**

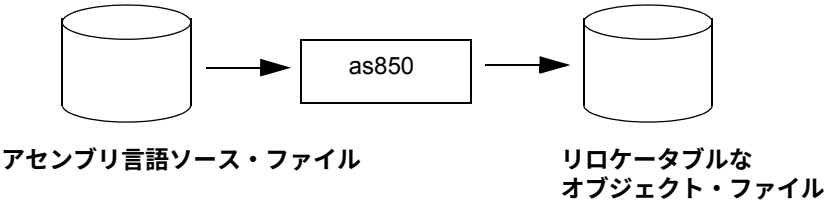
第 4 章　アセンブラ

この章では，アセンブラ（as850）の概要，操作方法，アセンブル・リストについて説明します。

4.1　動作の流れ

as850 は，指定されたアセンブリ言語ソース・ファイルをアセンブルし，リロケータブルなオブジェクト・ファイルを生成します（図 4 － 1 参照）。

図 4 － 1　as850 における動作の流れ



4.2 入出力ファイル

as850 では，次に示したファイルを入力ファイルとして指定できます。

- *file*.s…アセンブリ言語ソース・ファイル（.s ファイルと呼ばれる）

as850 によって生成されるリロケータブルなオブジェクト・ファイルは，.s を .o で置き換えたファイル名となります。

なお，ファイル名は Windows で認められるものであれば指定できますが，‘ @ ’ はコマンド・オプションと判断されるため ‘ @ ’ をファイル名の先頭に使用できません。また，ファイル／フォルダ名にスペースが含まれるものも使用できません。さらに，ファイルの漢字コードが EUC の場合，ファイル／フォルダ名に日本語は使用できません。

as850 によって生成されるリロケータブルなオブジェクト・ファイルは，その中に未解決な外部参照を含んでいる場合，それに対するリロケーションは未解決のままとなっています。

すべてのリロケーションを解決した実行可能なオブジェクト・ファイル（実行形式と呼ばれる）は，このリロケータブルなオブジェクト・ファイルをリンカ（ld850）でリンクすることによって生成されます。

4.3 操作方法

この節では，as850 の操作方法について説明します。

4.3.1 コマンド入力による方法

as850 は，ca850 コマンドからデフォルトで起動されますが，次の形式により単独でも起動できます。

コマンドは，コマンド・プロンプトで次のように入力します。

```
as850 [ オプション ]... ファイル名
      [ ]    ：[ ] 内は省略できます。
       ...   ：直前の [ ] 内のパターンの繰り返しができます。
```

4.3.2 PM+ による方法

アセンブリ言語ソース・ファイルに対する [アセンブラオプションの設定] ダイアログは，PM+ でプロジェクトを設定後，次のいずれかの操作で表示します。

- 対象プロジェクト全体のアセンブリ言語ソース・ファイルに設定する場合

(1) [ツール] メニュー→ [アセンブラオプションの設定] を選択

- 個別のアセンブリ言語ソース・ファイルに設定する場合

(1) PM+ のプロジェクト・ウインドウに表示された，プロジェクト・グループ名，またはプロジェクト・タイトル名をダブル・クリック
(2) “ ソース・ファイル ” をダブル・クリック
(3) オプションを設定したいソース・ファイル名を選択して右クリック
(4) [個別アセンブラオプションの設定] を選択

設定は次のようになります。

- タイトル選択状態：全体オプション設定
- “ ソース・ファイル ” フォルダを選択状態：全体オプション設定
- アセンブリ言語ソース・ファイルを選択状態：個別オプション設定

4.4 オプションの種類と機能

この節では，as850 のオプションを示します。

ただし，次の as850 のオプションで示すオプションについては，ca850 からそのまま as850 に渡すためには，ca850 に “ -Wa ” の指定が必要となります（「3.5.11 各モジュール」参照）。また，as850 のオプションには，PM+ 上では設定できないオプションが存在します。そのようなオプション指定が必要な場合は，コマンド・ラインから as850 を起動してください。

〔オプション説明でのマーク〕

【V850E2】	V850E2 コア専用のオプション
【V850E】	V850Ex コア専用のオプション
【PM+】	PM+ で指定項目が存在するオプション

4.4.1 ファイル

アセンブリ言語ソース・ファイルに対する前処理のオプションを指定します。

-a

【PM+】

アセンブル・リストを生成します。

ただし，-O オプション（最適化オプション）と同時に指定された場合，命令の並び替えにより，アセンブル・リストの出力が，一部不正確になる場合があります。

+err_file=*file*

エラー・メッセージをファイル *file* に追加保存します。

-err_file=*file*

エラー・メッセージをファイル *file* に上書き保存します。

-l *file*

【PM+】

-a オプションが指定された場合に生成されるアセンブル・リストをファイルに入れ，*file* をそのファイルの名前とします。-a オプションが指定されなかった場合は無効となります。このオプションを省略し，-a オプションが指定された場合，生成されるアセンブル・リストを標準出力に出力します。

4.4.2 オプション

アセンブリ言語ソース・ファイルに対する as850 のオプションを指定します。

-D*name*[=*def*]

【PM+】

定義したいマクロ名を指定します。“ =*def* ” の部分を省略した場合，def は 1 とみなします。アセンブリ言語ソース・プログラムの前に，.set *name*, *def* が記述されたものとみなします。

-G*num*

【PM+】

外部ラベルへのアクセスに対し，“ *num* バイト以下のサイズのデータはすべて sdata 属性セクション，または sbss 属性セクションに割り付けられる ” ことを想定した機械語命令を生成します。

num は 10 進数で 0 ～ 32,767 の範囲の整数が指定可能です。このオプションを省略した場合，*num*= ∞とみなします。

ただし，疑似命令 “ .option sdata ” で sdata が指定されたデータは，そのサイズに関係なく sdata 属性／sbss 属性セクションに割り付けられることを想定したアセンブラ命令を生成します。

なお，ca850 から起動された場合，ca850 の起動時に指定された -G*num* が渡されます。

-I *dir*

【PM+】

ファイル入力疑似命令（.include ／ .binclude）で指定したファイルを，ソース・ファイルの置かれているフォルダに優先して検索するフォルダとして指定します。指定したフォルダを検索しても見つからない場合，およびこのオプションを省略した場合，ソース・ファイルの置かれたフォルダ，C 言語ソース・ファイルの置かれたフォルダ，カレント・フォルダの順で検索します。

-m

【PM+】

マスク・レジスタ機能を使用するという情報を持つオブジェクト・ファイルを作成します。この機能を使用すると，as850 は r20 に 8 ビットのマスク値 0xff，r21 に 16 ビットのマスク値 0xffff が設定されているものとしてコードを出力します。マスク・レジスタ（r20，r21）へのマスク値の設定は，スタート・アップ・ルーチン等，ユーザ・プログラムで行う必要があります。マスク・レジスタ機能を利用するかどうかの判断には，利用する側で次の点を十分考慮する必要があります。

- マスク・コードが多く出力されるプログラムであるか。
- マスク・レジスタとして使用するため，レジスタ変数用レジスタが 2 本少なくなるが，その影響はないか。

-O

【PM+】

命令を並べ替えて，レジスタ／フラグのハザードを回避する最適化を行います。-g オプション（デバッガ用の情報出力）と同時に指定した場合，このオプションは無視され，-g オプションが有効となります。また，V850 コアのターゲット・デバイス指定，または V850 コア共通オブジェクト作成時に -p オプション（CPU の不具合回避オプション）と同時に指定した場合も，このオプションは無視され，-p オプションが有効となります。

V850Ex ／ V850E2 コアのターゲット・デバイス指定，または V850Ex ／ V850E2 コア共通オブジェクト作

成時に -p オプションと同時に指定した場合，このオプション，および -p オプションはともに有効となります。

-v

【PM+】

as850 の実行状況の詳細を標準エラー出力に出力します。

-w

【PM+】

次の場合に対し，警告メッセージを出力しません。

(1) r1 をソース・レジスタ，またはディスティネーション・レジスタとして指定

(2) r0 をディスティネーション・レジスタとして指定

(3) マスク・レジスタ機能使用時に r20，または r21 をディスティネーション・レジスタとして指定

-w*string*+

-w*string*-

【PM+】

-w オプションの指定にかかわりなく，項目ごとに警告メッセージの出力，および抑止を指定します。“ + ” を付けた場合，メッセージを出力し，“ - ” を付けた場合，メッセージを抑止します。

string に指定できる文字列は，次のとおりです。

r0	r0 をデスティネーション・レジスタとして指定
r1	r1 をソース・レジスタ，またはデスティネーション・レジスタとして指定

なお，“ + ”，および “ - ” を付けない場合，エラーとなります。

-Xfar_jump

【V850E2】【PM+】

アセンブラに，V850E2 コアのデバイスを品種指定している場合に，命令に 22/32 を記述しない分岐命令（jarl，jr）に対して，far jump にするように指定します。命令単位で変更する場合には，jarl22 / jarl32，または，jr22 / jr32 と明示したものを記述してください。

また，jmp 命令に関しては，-Xfar_jump オプションの影響を受けません。

4.4.3 デバイス

アセンブリ言語ソース・ファイルに対する as850 のデバイスに関するオプションを指定します。

-X256M

【V850E】【PM+】

メモリ空間を 256M バイトとして扱います。このオプションを指定しない場合はメモリ空間を 64M バイトとして扱い，アドレス解決します。このオプションは使用するチップセットにあわせて設定してください。

V850Ex コアでは，物理アドレス空間が 256M バイト持つ場合が多く，64M を越えた 256M までの空間を使用するアプリケーションを作成する場合，このオプションを指定してください。

-bpc=*num*

【PM+】

プログラマブル周辺 I/O レジスタの上位アドレスを設定します。*num* には，BPC レジスタの最上位ビットを除いたアドレス部分のみを指定してください。ターゲット・デバイスがプログラマブル周辺 I/O レジスタ機能を持ち（V850E/IA1 など），変更可能なアドレス部分（=BPC レジスタに設定する値）を設定したい場合，アプリケーションのアセンブル時に，値を確定させる必要があります。そこでこのオプションを指定すると，指定した値を使用してアセンブルします。このオプション指定時には必ず値を指定してください。値には 2 進数，8 進数，10 進数，16 進数を使用できます。不正な値を指定した場合，および BPC レジスタに設定可能な範囲を越える値を指定した場合，警告メッセージが出力され，このオプションは無視されます。

例

```
-bpc=0x1234
```

上記の場合，ターゲット・デバイスが V850E/IA1 の場合，プログラマブル周辺 I/O レジスタ領域の先頭アドレスは，この値を 14 ビット左シフトした 0x48d0000 として扱われます。

設定する値はアプリケーション全体で１つです。ファイルごとのオプション設定で“ -Xbpc ”や“ -bpc ”を指定する場合，ファイル間で同じ値にしてください。

ただし，プログラマブル周辺 I/O レジスタを使用しないファイルに対しては，このオプションを指定する必要はありません。

プログラマブル周辺 I/O レジスタ機能を持たないターゲット・デバイスの場合，および V850 コア，V850Ex コア共通としてアセンブルする場合，このオプションを指定すると，警告メッセージが出力され，このオプションは無視されます。

なお，このオプションは，プログラマブル周辺 I/O レジスタのアドレスを，アセンブル時に確定するためのものであり，BPC レジスタに実際に値を反映させるものではありません。動作のためには，別途，スタート・アップ・モジュールなどで BPC レジスタに値を設定する必要があります。

“ CA850 ユーザーズ・マニュアル C 言語編 ”にスタート・アップ・ルーチンのサンプルが載っていますので，そちらを参照してください。また，パッケージに含まれるスタート・アップ・モジュールにも，サンプルが載っています（コメントアウトしてあります）。

例

V850E/IA1 **で，プログラマブル周辺** I/O **レジスタの先頭アドレスの変更可能部分を“** 0x1234 **”とし，この機能の使用を許可するフラグ** 0x8000 **を設定する場合，次の記述をスタート・アップ・モジュールに記述します。**

```
mov    0x9234,r10    -- 0x1234 | 0x8000 = 0x9234
st.h   r10, BPC
```

as850 **は，このオプションを指定するか，省略してもプログラマブル周辺** I/O **レジスタを参照している場合，特殊な予約セクションの** .bpc **セクションを出力します。このセクションは，リンク時のチェックのために使用されます。**.bpc **セクションは，情報用の特殊な予約セクションであり，メモリにロードされることはありません。したがって，通常のセクションのように，リンク・ディレクティブに記述する必要はありません。**

4.4.4 その他

アセンブリ言語ソース・ファイルに対する as850 のその他のオプションを設定します。

-cn

生成するオブジェクトに，マジック・ナンバとして，V850 コア共通の“ 共通マジック・ナンバ ”の値が埋め込まれます。これにより，V850 コア内で共通に使用可能なオブジェクトとなります。このオプションを省略した場合，指定されたターゲット・デバイスに定義されたマジック・ナンバが埋め込まれます。

-cnv850e

【V850E】

生成するオブジェクトのマジック・ナンバとして，V850Ex コア共通の“ 共通マジック・ナンバ ”の値を設定します。これにより，V850Ex コア内で共通に使用できるオブジェクトとなります。このオプションを省略した場合，指定されたターゲット・デバイスに定義されたマジック・ナンバを設定します。

-cnv850e2

【V850E2】

生成するオブジェクトのマジック・ナンバとして，V850E2 コア共通の“ 共通マジック・ナンバ ” の値を設定します。これにより，V850E2 コア内で共通に使用できるオブジェクトとなります。このオプションを省略した場合，指定されたターゲット・デバイスに定義されたマジック・ナンバを設定します。

-cpu *devicename*

ターゲット・デバイスを指定します。このオプション指定は疑似命令“ .option cpu ”より優先されます。このオプション，または疑似命令“ .option cpu ”によるターゲット・デバイスの指定と -cn ／ -cnv850e ／ -cnv850e2 指定の両方がある場合は，ターゲット・デバイス固有の情報を含むコア共通オブジェクトを作成できます。

疑似命令“ .option cpu ”による指定，または -cn ／ -cnv850e ／ -cnv850e2 オプション指定がない場合，このオプションを省略するとアセンブルを中止します。

-F *devpath*

デバイス・ファイルの置かれているフォルダを指定します。このオプションを省略した場合，標準フォルダを指定します。

-f

“ 新形式の関数呼び出しである ”という情報を持つオブジェクト・ファイルを作成します。旧バージョンの CA850（Ver.1.xx）で作成されたアセンブリ言語ソース・ファイルを使用する場合に有効です。現バージョンでは，デフォルトでこのオプションが指定されたものとしてアセンブルします。

-g

【PM+】

デバッグ情報を出力します。デバッガでアセンブリ言語ソース・デバッグを行いたい場合など，プログラムをデバッグする際に指定してください。

なお，最適化（-O）オプションを同時に指定した場合，ソース・ファイル中にデバッグ情報用セクションがあれば，このオプションは無視されます。デバッグ情報用のセクションがなければ，最適化（-O）が無視され，このオプションが有効になります。つまり，デバッグ情報がなければ，このオプションが優先されます。

-o *ofile*

アセンブルして出力するオブジェクト・ファイル名を *ofile* とします。このオプションを省略した場合，オブジェクト・ファイル名は，ソース・ファイルの拡張子 .s を .o に置き換えたものとなります。

-p[*num*]

CPU の不具合を回避するためのコードを出力します。*num* には，出力するコードの種類（1 ～ 10，および 4a）を指定します。1 ～ 4 と 4a は V850 コアに，5 ～ 10 は V850Ex ／ V850E2 コアに有効です。*num* を省略した場合，デバイス・ファイルから判断し，次の種類のコードを出力します。

(1) ターゲット・デバイスが V850Ex コア，またはアセンブラ・オプションで [マジックナンバ] に“ V850Ex コア共通 [-cnv850e] ” を指定した場合，5 ～ 10 のコードを出力。
ターゲット・デバイスが V850E2 コア，またはアセンブラ・オプションで [マジックナンバ] に “ V850E2 コア共通 [-cnv850e2] ” を指定した場合，5 ～ 10 のコードを出力。

(2) ターゲット・デバイスが V850 デバイスの場合，1 ～ 3 のコードを出力。

(3) アセンブラ・オプションで [マジックナンバ] に “ V850 コア共通 [-cn] ” を指定した場合，1 ～ 3，5 ～ 10 のコードを出力。

このオプションにより出力されるコードについての詳細は「4.7.2 CPU 不具合の回避オプション」を参照してください。

-v

as850 のバージョン情報を標準エラー出力に出力し，終了します。

-zf

フラッシュ／外付け ROM 再リンク機能を使用した .ext_func 疑似命令の記述があるアセンブリ言語ソース・ファイルに対し，フラッシュ／外付け ROM 側のアセンブル処理を行います。このオプションを省略した場合，フラッシュ／外付け ROM 再リンク機能を使用したアセンブリ言語ソース・ファイルに対し，ブート／内蔵 ROM 側のアセンブル処理を行います。

なお，フラッシュ／外付け ROM 再リンク機能を使用しないアセンブリ言語ソース・ファイルに対し，このオプションを指定する必要はありません。指定した場合，機能が変化することはありません。また，警告メッセージを出力しません。

フラッシュ／外付け ROM 再リンク機能の詳細は「5.6 フラッシュ／外付け ROM 再リンク機能」を参照してください。

@*cfile*

cfile をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

4.5 PM+ での設定

この節では，対象プロジェクトのソース・ファイルに対して，as850 のコマンド・オプションを設定するダイアログについて説明します。

[アセンブラオプションの設定] ダイアログは，[プロジェクト] メニューからプロジェクトを設定後，次のいずれかの操作で表示します。

- 対象プロジェクト全体のアセンブリ言語ソース・ファイルに設定する場合

(1) [ツール] メニュー→ [アセンブラオプションの設定] を選択

- 個別のアセンブリ言語ソース・ファイルに設定する場合

(1) PM+ のプロジェクト・ウインドウに表示された，プロジェクト・グループ名，またはプロジェクト・タイトル名をダブル・クリック
(2) “ ソース・ファイル ” をダブル・クリック
(3) オプションを設定したいソース・ファイル名を選択して右クリック
(4) [個別アセンブラオプションの設定] を選択

個別の設定が行われている場合，そのファイルに対しては，全体のオプション設定が無効になります。個別オプションが設定されたファイルには，プロジェクト・ウインドウに表示される “ ソース・ファイル ” のファイル名の先頭にあるアイコンが緑色に変化します。

個別のオプション設定を無効にし，全体のオプション設定を有効にしたい場合は，そのファイルを選択して右クリックで表示される [個別アセンブラオプションの解除],または [アセンブラオプションの設定] ダイアログの [オプション] タブと [差分] タブに追加されている [ソースオプションの削除] ボタンをクリックします。これにより個別の設定が削除されます。

なお，ダイアログ上の “ [] ” 内に表示されたオプション名は，コマンド・プロンプトから起動する場合のオプション名です。

また，出力オブジェクト名を指定したい場合など，コマンド・プロンプトからの起動時にのみ指定可能なオプションもあります。

フラッシュ領域の再リンク機能を使用する場合，フラッシュ領域側のオブジェクト生成には，as850 のオプション “ -zf ” が必要です。PM+ からのオプション設定時には，[コンパイラ共通オプションの設定] ダイアログの [フラッシュ] タブの “ フラッシュ対応オブジェクトの生成 ”，または “ フラッシュ側アーカイブファイルの生成 ” で設定します。

4.5.1 [アセンブラオプションの設定] ダイアログ

[アセンブラオプションの設定] ダイアログの上部には，次の 1 つのタブが表示されており，タブの選択により，ダイアログの表示内容が変わります。

表 4 － 1 [アセンブラオプションの設定] ダイアログ

タブ	内容
[オプション]	アセンブラのオプションの設定

なお，個別のアセンブリ言語ソース・ファイルへのオプション指定場合は，次の 2 個のタブになります。

表 4 － 2 [アセンブラオプションの設定ダイアログ]（ソース個別）

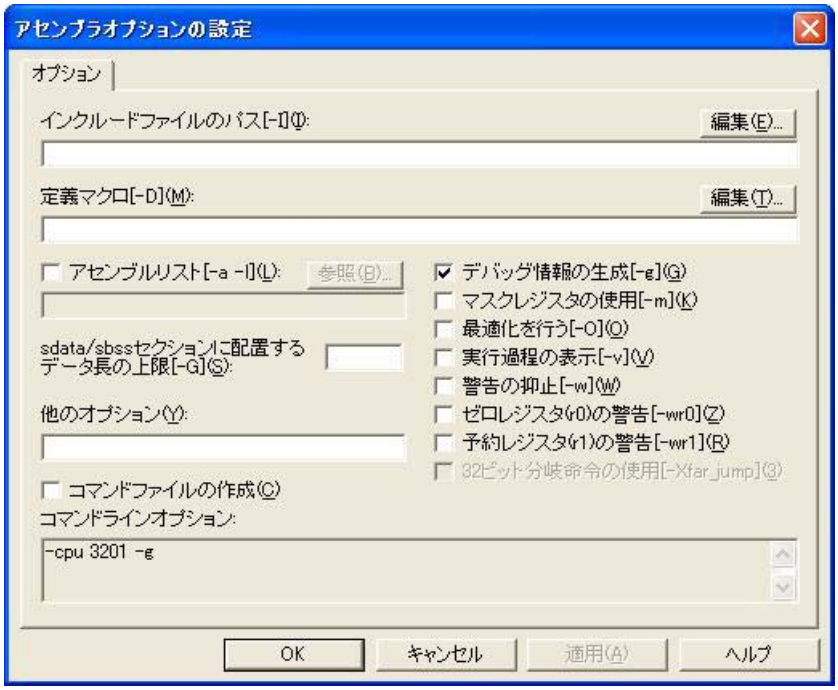
タブ	内容
[オプション]	アセンブラのオプションの設定
[差分]	プロジェクト全体オプションとソース個別オプションの差分を表示

なお，ダイアログ上の “ [] ” 内に表示されたオプション名は，コマンド・プロンプトから起動する場合のオプション名です。

[オプション]

アセンブラのオプションの設定を行います。

図 4 － 2 [アセンブラオプションの設定] ダイアログ：[オプション] タブ



(1) インクルードファイルのパス [-I]

ファイル入力疑似命令（.binclude ／ .include）で指定したファイルを，ソース・ファイルの置かれているフォルダに優先して検索するフォルダを指定します。複数のパスを指定する場合，“ ; （セミコロン）” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，パス項目をダイアログ上で編集することができます。

なお，このオプションは，ソース個別の指定では選択できません。常に全体のオプションとして設定します。このオプションを省略した場合，および指定したフォルダで見つからない場合，次の順に検索します。

(a) アセンブリ言語ソース・ファイルの置かれているフォルダ

(b) C 言語ソース・ファイルから作成されたアセンブリ言語ソース・ファイルの場合，元になった C 言語ソース・ファイルの置かれているフォルダ（.file 疑似命令により検出）

(c) プロジェクト・フォルダ（コマンド・プロンプトからのコマンド起動の場合，コマンド起動時のカレント・フォルダ）

(2) 定義マクロ [-D]

定義したいマクロ名を “ マクロ名 = 定義値 ” の形式で指定します。“ = 定義値 ” の部分を省略した場合，定義値は 1 とみなします。アセンブリ言語プログラムの前に “ .set マクロ名 , 定義値 ” が記述されたものとみなします。複数のマクロを定義する場合，“ ; （セミコロン）” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，定義マクロ項目をダイアログ上で編集することができます。

(3) アセンブルリスト [-a -l]

アセンブリ言語ソースのアセンブル結果を，アセンブル・リストに出力するかどうかを設定するチェック・ボックスです。チェックされている場合は，下のエディット・ボックスにフォルダ名やファイル名が指定可能になります。このエディット・ボックスに“全体オプションの設定”として“フォルダ名”を指定し，そのフォルダが存在しなかった場合は，フォルダを作成するかどうかを確認するメッセージ・ボックスが表示されます。エディット・ボックスに何も指定しなかった場合は，アセンブリ言語ソース・ファイル名の拡張子を .v として，プロジェクト・フォルダにアセンブル・リストを出力します。別の出力先を指定したい場合は，エディット・ボックスにフォルダ名を指定してください。

ファイル名を指定する場合ですが“全体オプションの設定”でファイル名を指定すると，同じファイル名で上書きするため，結果的に最後にアセンブルしたソースに対するアセンブル・リストが出力されます。ファイル名を指定するときは，“ソース個別オプション設定”でファイル名を指定する場合に有効になります。

(4) sdata ／ sbss セクションに配置するデータ長の上限 [-G]

.sdata ／ .sbss セクションに割り当てるデータ長の上限を指定します。指定したサイズ（バイト）以下のデータを .sdata セクション，または .sbss セクションに割り付けます。

ただし，疑似命令“.option sdata”で，sdata が指定されたデータは，そのサイズに関係なく，sdata 属性／sbss 属性セクションに割り付けられることを想定したアセンブラ命令を生成します。

指定できる値は 10 進数で 0 ～ 32,767 の範囲の整数が指定可能です。ここで指定する値の目安は，[リンカオプションの設定] ダイアログの [オプション] タブの“GP 情報の表示 [-A]”を指定することで出力されるので，参考にしてください。

なお，このオプション指定は，[コンパイラオプションの設定] ダイアログと共有されます。

指定した値が，そのまま [コンパイラオプションの設定] ダイアログの [出力コード] タブの“sdata ／ sbss セクションに配置するデータ長の上限 [-G]”に設定されます。また，このオプションは，ソース個別の指定では選択できません。常に全体のオプションとして設定します。

.sdata セクション，.sbss セクションについての詳細は“CA850 ユーザーズ・マニュアル C 言語編”を参照してください。

(5) 他のオプション

前記の“アセンブラオプションの設定”では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

現在のところ，“他のオプション”で指定できるオプションは次のとおりです。

- -p

これ以外のオプションを指定することもできますが，現状未サポートです。

(6) マスクレジスタの使用 [-m]

マスク・レジスタ機能を使用するという情報を持つオブジェクト・ファイルを作成します。この機能を使用すると，as850 は r20 に 8 ビットのマスク値 0xff，r21 に 16 ビットのマスク値 0xffff が設定されているものとしてコードを出力します。マスク・レジスタ（r20，r21）へのマスク値の設定は，スタート・アップ・ルーチン等，ユーザ・プログラムで行う必要があります。マスク・レジスタ機能を利用するかどうかの判断には，利用する側で次の点を十分考慮する必要があります。

- マスク・コードが多く出力されるプログラムであるか。
- マスク・レジスタとして使用するため，レジスタ変数用レジスタが 2 本少なくなるが，その影響はないか。

なお，このオプションを指定したとき，マスク・レジスタを使用したオブジェクトとしないオブジェクトが混在する場合，ld850 でエラーとなります。

(7) 最適化を行う [-O]

命令を並べ替えて，レジスタ／フラグのハザードを回避する最適化を行います。-g オプション（デバッガ用の情報出力）と同時に指定した場合，このオプションは無視され，-g オプションが有効となります。また，V850 コアのターゲット・デバイス指定，または V850 コア共通オブジェクト作成時に -p オプション（CPU の不具合回避オプション）と同時に指定した場合も，このオプションは無視され，-p オプションが有効となります。

V850Ex ／ V850E2 コアのターゲット・デバイス指定，または V850Ex ／ V850E2 コア共通オブジェクト作成時に -p オプションと同時に指定した場合，このオプション，および -p オプションはともに有効となります。

(8) 実行過程の表示 [-v]

as850 の実行状況の詳細をアウトプット・ウインドウに表示します。

(9) デバッグ情報の生成 [-g]

デバッグ情報を出力します。デバッガでアセンブリ言語ソース・デバッグを行いたい場合など，プログラムをデバッグする際にはチェックしてください。なお，最適化（-O）オプションを同時に指定した場合，ソース・ファイル中にデバッグ情報用セクションがあれば，このオプションは無視されます。デバッグ情報用のセクションがなければ，最適化（-O）が無視され，このオプションが有効になります。つまり，デバッグ情報がなければ，このオプションが優先されます。

(10) 警告の抑止 [-w]

-w オプションで指定する警告メッセージを抑止するかどうかを指定します。チェックされている場合は，-w オプションを指定したのと同じになります。

(11) ゼロレジスタ（r0）の警告 [-wr0]

-wr0 オプションで指定する r0 レジスタをデスティネーション・レジスタに使用した場合の警告メッセージを抑止するかどうかを指定します。チェックされている場合は，-wr0+ オプションを指定したのと同じになります。また，グレーの場合は，-wr0- オプションを指定したのと同じになります。

(12) 予約レジスタ（r1）の警告 [-wr1]

-wr1 オプションで指定する r1 レジスタを使用した場合の警告メッセージを抑止するかどうかを指定します。チェックされている場合は，-wr1+ オプションを指定したのと同じになります。また，グレーの場合は，-wr1- オプションを指定したのと同じになります。

(13) 32 ビット分岐命令の使用 [-Xfar_jump]【V850E2】

アセンブラに，V850E2 コアのデバイスを品種指定している場合に，命令に 22/32 を記述しない分岐命令（jarl，jr）に対して，far jump にするように指定します。命令単位で変更する場合には，jarl22 / jarl32，または，jr22 / jr32 と明示したものを記述してください。

また，jmp 命令に関しては，-Xfar_jump オプションの影響を受けません。

(14) コマンドファイルの作成

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。チェックされている場合は，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，文字列の制限を意識する必要がなくなります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。なお，このオプションはデフォルトではチェックされていません。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

(15) コマンドラインオプション

ダイアログで設定したオプションをコマンド・ライン・オプションで表示します。

ただし，出力ファイル名を設定する -a -l オプションは表示されません。

なお，このエリアは参照用のため，書き込みはできません。

[備考]

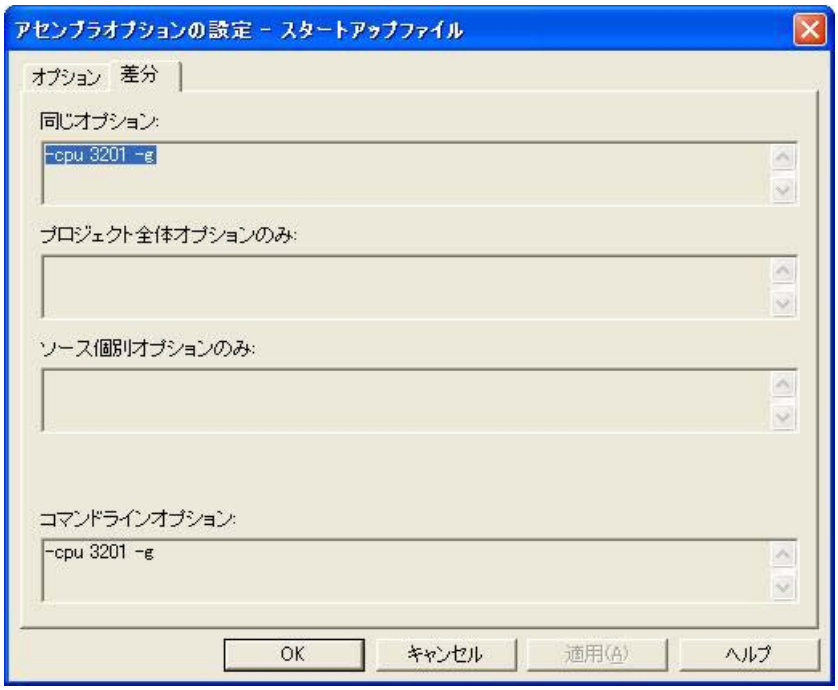
(a) [ソースオプションの削除] ボタン

ソース・ファイル個別のオプション設定を行っている場合に選択可能なボタンです。プロジェクト全体のオプション指定時には表示されず，選択できません。個別オプションを無効にし，全体のオプション設定を有効にしたい場合は，このボタンをクリックします。

[差分]

プロジェクト全体オプションとソース個別オプションの差分を表示します。

図４－３ [アセンブラオプションの設定] ダイアログ：[差分] タブ



(1) 同じオプション

プロジェクト全体オプションとソース個別オプションの双方で指定されているオプションがコマンド・ライン形式で表示されます。

なお，このエリアは参照用のため，書き込みはできません。

(2) プロジェクト全体オプションのみ

プロジェクト全体オプションで指定されていてソース個別オプションでは指定されていないオプションがコマンド・ライン形式で表示されます。

なお，このエリアは参照用のため，書き込みはできません。

(3) ソース個別オプションのみ

ソース個別オプションで指定されていてプロジェクト全体オプションでは指定されていないオプションがコマンドライン形式で表示されます。

なお，このエリアは参照用のため，書き込みはできません。

(4) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[備考]

(a) [ソースオプションの削除] ボタン

ソース・ファイル個別のオプション設定を行っている場合に選択可能なボタンです。プロジェクト全体のオプション指定時には表示されず，選択できません。個別オプションを無効にし，全体のオプション設定を有効にしたい場合は，このボタンをクリックします。

4.6 アセンブル・リスト

この節では，アセンブル・リストについて説明します。アセンブル・リストとは，ソースをコンパイル，アセンブルして出力されるコードをリスト形式にしたものです。これによってコンパイル，アセンブルした結果が，どのようなコードになっているかを確認できます。

4.6.1 出力方法

アセンブル・リストの出力方法は次のとおりです。

(1) コマンド入力の場合

-a オプションを指定すると，アセンブル・リストを標準出力に出力します。-a オプションと同時に -l オプションで出力ファイル名を指定すると，そのファイルにアセンブル・リストを出力します。

ca850 で C 言語ソース・ファイルのコンパイル時に「アセンブル・リストの出力」を指定し，-Xc オプションで「ソースのコメントの出力」を指定すると，アセンブル・リストには，そのコードに対応した C 言語ソース行をコメント表示します。

ただし，最適化を強力にした場合，コード行とソース行が一致しないことがあります。

(2) PM+ の場合

[アセンブラオプションの設定] ダイアログで“ アセンブルリスト [-a -l] ”を指定します。リストはファイルに出力され，そのファイル名は，拡張子を .v で置き換えたものとなります。

C 言語ソース・ファイルのコンパイル時に [コンパイラオプションの設定] ダイアログの [出力ファイル] タブの“ アセンブルリスト [-Fv] ”を指定し，さらに [出力コード] タブの“ ソースのコメントの出力 [-Xc] ”を指定すると，アセンブル・リストには，そのコードに対応した C 言語ソース行をコメント表示します。

ただし，最適化を強力にした場合，コード行とソース行が一致しないことがあります。

4.6.2 出力例

次にアセンブル・リストの出力例を示します。例にある C 言語ソース・ファイルをコンパイルし，さらに出力されたアセンブリ言語ソース・ファイルをアセンブルすることにより出力されたアセンブル・リストの一例を次に示します。

C 言語ソース・ファイル

```
void main(void)
{
        int     a;
}
```

アセンブル・リストの出力

```
                        ...
A-X- 00000000                41              .file  "c:\work\src\a.c"
A-X- 00000000                42              .align 4
A-X- 00000000                43              #@BF
A-X- 00000000                44              .frame _main, .S2
A-X- 00000000                45              .globl _main
A-X- 00000000                46      _main:
A-X- 00000000                47              #@B_PROLOGUE
A-X- 00000000 D505           48              jbr    .L15
A-X- 00000002                49      .L16:
A-X- 00000002                50      .G17:
A-X- 00000002                51      .G18:
A-X- 00000002                52      .G9:
A-X- 00000002                53      .G11:
A-X- 00000002                54      .G19:
A-X- 00000002                55              #@B_EPILOGUE
A-X- 00000002 23FF0100               56              ld.w   -4+.F2[sp], lp
A-X- 00000006 441A           57              add    .S2, sp
A-X- 00000008 7F00           58              jmp    [lp]   --0
A-X- 0000000A                59              #@E_EPILOGUE
A-X- 0000000A                60      .L15:
A-X- 0000000A 5C1A           61              add    -.S2, sp
A-X- 0000000C 63FF0100               62              st.w   lp, -4+.F2[sp]
A-X- 00000010                63              #@E_PROLOGUE
A-X- 00000010 95F0           64              jbr    .L16
A-X- 00000012                65              #@FUNC_ARG
A-X- 00000012                66      .G5:
A-X- 00000012                67              .set   .S2, 0x4
A-X- 00000012                68              .set   .F2, 0x4
A-X- 00000012                69              .set   .A2, 0x0
A-X- 00000012                70              .set   .T2, 0x0
A-X- 00000012                71              .set   .P2, 0x0
A-X- 00000012                72              .set   .R2, 0x0
A-X- 00000012                73              .set   .X2, 0x0
                        ...

(1)     (2)         (3)         (4)                 (5)
```

アセンブル・リストの意味は，次のようになります。

(1) セクション属性

その行が格納されるセクションのセクション属性です。セクション属性とその意味は次のようになります。

表 4 － 3 セクション属性とその意味

セクション属性	意味
A	メモリを占有するセクション
W	書き込み可能なセクション
X	実行可能なセクション
G	グローバル・ポインタ（gp）と 16 ビットのディスプレースメントを用いて参照することのできるメモリの範囲に割り付けるセクション

(2) ロケーション・カウンタ値

コードの先頭に対するロケーション・カウンタ値です。

(3) コード

生成されたコードです。2 桁の 16 進数で表記されます。

(4) 行番号

その行の行番号です。10 進数で表記されます。

(5) ソース・プログラム

その行のアセンブリ言語ソース・プログラムです。その行の命令に対して命令展開が生じた場合，その命令展開によって生成された命令列を “ -- ” 以降に示します。また，その行のアセンブリ言語ソース・プログラムに対する C 言語ソース・プログラムも，このエリアに表示されます。

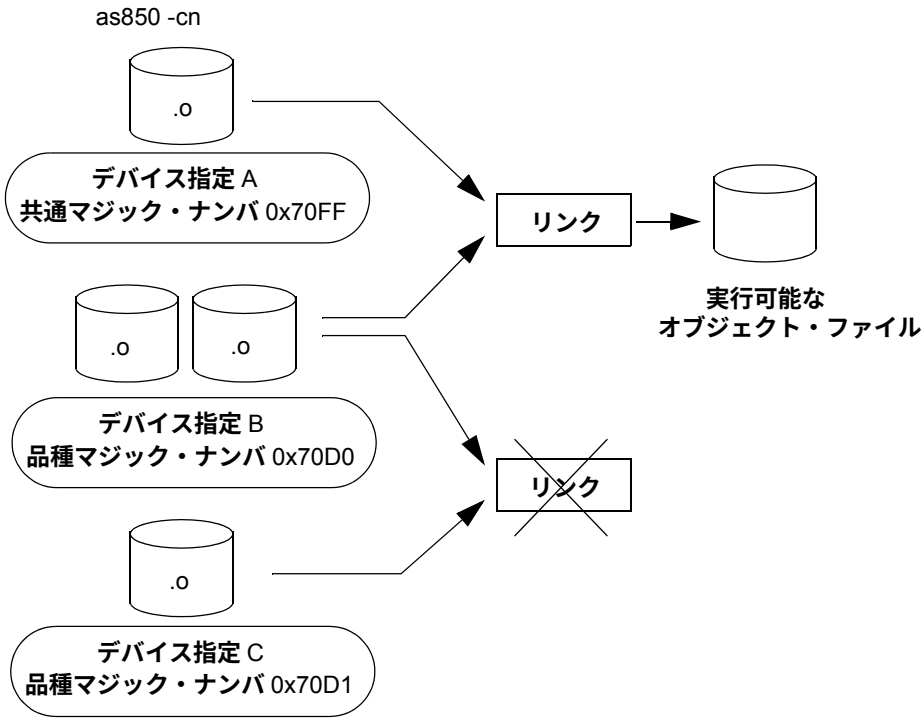
4.7 注意事項

4.7.1 マジック・ナンバ

as850 が生成するオブジェクトには，そのオブジェクトが対象としているデバイス情報が自動的に埋め込まれます。この情報を “ マジック・ナンバ ” と呼びます。オブジェクトが，あるデバイス品種のみを対象としている場合，「品種固有のマジック・ナンバ」が埋め込まれ，コア全体を対象としている場合，「共通のマジック・ナンバ」が埋め込まれます。

as850 で -cn オプションを指定してアセンブルされたオブジェクトは，共通マジック・ナンバを持つため，同じコア内であれば，異なるデバイス品種を指定されたオブジェクトとリンク可能です（ld850 によるリンク時にエラーとなりません）。このため，-cn オプションを指定して生成したオブジェクトは，コア内で共通して使用可能なオブジェクトとなります。

図 4 － 4 as850 における共通オブジェクト作成イメージ

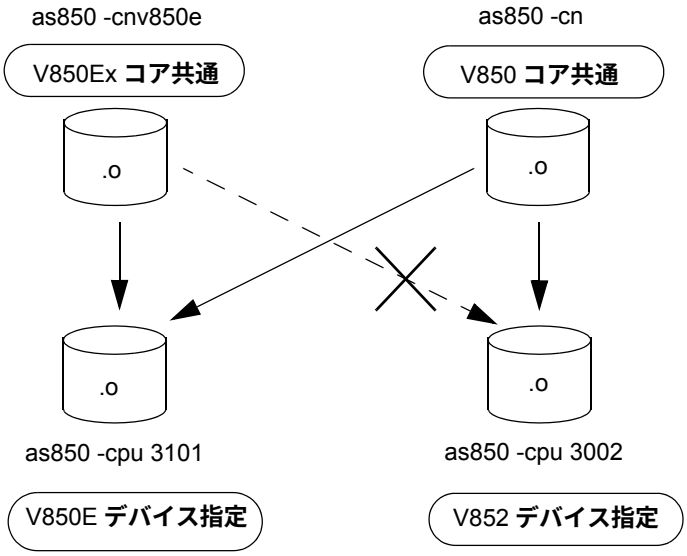


[注意事項]

- **コア共通のマジック・ナンバと，品種固有のマジック・ナンバは，各デバイス・ファイルに定義され，対応付けられています。**as850 **はデバイス・ファイルを参照し，マジック・ナンバを埋め込みます。**
- **品種固有の周辺機能レジスタなどを操作するオブジェクト・ファイルは，コア共通にしないでください。**
- -cpu **オプション，または** .option **疑似命令によってターゲット・デバイスを指定したうえで，**-cn ／ -cnv850e ／ -cnv850e2 **オプションによる指定を行うと，ターゲット・デバイス固有の情報を含むコア共通オブジェクトが作成できます。**

  ただし，ターゲット・デバイスと異なるデバイス固有情報を持たせたオブジェクトは正常に動作しないので，目的のターゲット・デバイスに流用可能なデバイス固有情報であることをあらかじめ確認しておいてください。
- V850Ex **コアは，**V850 **コアに対して上位互換です。**V850 **コアで使用していたソース・ファイルを，**V850Ex **コアで使用できます。その場合，“** -cn **”，または “** -cnv850e **” オプションを指定してオブジェクトを作成してください。“** -cn **” で作成された** V850 **コア共通オブジェクトは，**V850Ex **コア・オブジェクトともリンク可能です。一方，“** -cnv850e **” で作成したオブジェクトは，**V850 **コア・オブジェクトとはリンクできません。**
- V850E2 **コアは，**V850 ／ V850Ex **コアに対して上位互換です。**V850 ／ V850Ex **コアで使用していたソース・ファイルを，**V850E2 **コアで使用できます。その場合，“** -cn **”，または “** -cnv850e **” オプションを指定してオブジェクトを作成してください。“** -cn **” で作成された** V850 ／ V850Ex **コア共通オブジェクトは，**V850E2 **コア・オブジェクトともリンク可能です。一方，“** -cnv850e **” で作成したオブジェクトは，**V850 ／ V850Ex **コア・オブジェクトとはリンクできません。**

図 4 － 5 as850 **における** CPU **コアによる互換関係の例（**V850Ex **コアと** V850 **コアの場合）**



4.7.2 CPU 不具合の回避オプション

CA850 では，V850 コア／V850Ex／V850E2 コア CPU の不具合を回避するため，ca850 において -Xv850patch オプション，as850 において -p オプションを提供しています。ca850 から as850 を起動する場合，ca850 の -Xv850patch オプションを指定すると，（ca850 が出力するアセンブリ言語ソース・ファイルに対して）as850 で同じ *num* を持つ -p オプションが自動的に設定されます。

num には，出力するコードの種類（1 ～ 10，および 4a）を指定します。1 ～ 4 と 4a は V850 コアに，5 ～ 10 は V850Ex ／ V850E2 コアに有効です。*num* を省略した場合，デバイス・ファイルから判断し，次の種類のコードを出力します。

- ターゲット・デバイスが V850E2 コア，またはアセンブラ・オプションで [マジックナンバ] に “ V850E2 コア共通 [-cnv850e2] ” を指定した場合，5 ～ 10 のコードを出力。
- ターゲット・デバイスが V850Ex コア，またはアセンブラ・オプションで [マジックナンバ] に “ V850Ex コア共通 [-cnv850e] ” を指定した場合，5 ～ 10 のコードを出力。
- ターゲット・デバイスが V850 コアの場合，1 ～ 4 と 4a のコードを出力。
- アセンブラ・オプションでマジックナンバに “ V850 コア共通 [-cn] ” を指定した場合，1 ～ 10，および 4a のコードを出力。

[注意事項]

- 使用している CPU に該当する不具合があるかどうかは，CPU 添付の資料を参照してください。
- V850 コアのターゲット・デバイス指定，または V850 コア共通オブジェクト作成時に -p オプションと as850 の最適化（-O）オプションを同時に指定した場合，-p が優先され，-O は無視されます。
- V850Ex ／ V850E2 コアのターゲット・デバイス指定，または V850Ex ／ V850E2 コア共通オブジェクト作成時に -p オプションと as850 の最適化（-O）オプションを同時に指定した場合，-p，および -O はともに有効になります。
- 不具合の発生するコード・パターンが，異なるセクションにまたがっている場合，このオプションの機能は無効です。
- -Xv850patch=11 オプションのみ ca850 で処理します。
- CPU コアと，回避オプションに対応する不具合は，次のとおりです（保守品と廃品種を除く，最新バージョンの μPD(F)703xxx の場合）。
  なお，使用している CPU に該当する障害であるかどうかは，CPU の資料を参照してください。

表 4 － 4　CPU コアと -p オプションに対応する不具合

CPU コア	-p1	-p2	-p3	-p4	-p4a	-p5	-p6	-p7	-p8	-p9	-p10
V850 コア	○	○	○	○	○	－	－	－	－	－	－
V850E/MS1	－	－	－	－	－	○	－	－	×	－	×
V850E1 コア	－	－	－	－	－	－	○	○	－	○	－
V850ES コア	－	－	－	－	－	－	－	－	－	－	－

備考 ×：該当
○：修正済み（保守品と廃品種を除く，最新バージョンの μPD(F)703xxx の場合）
－：非該当

num の種類と意味は，次のとおりです。

なお，命令，およびレジスタについては，各デバイスの“ユーザーズ・マニュアル アーキテクチャ編”を参照してください。

(1) 1（-Xv850patch=1 → -p1）

“ld.w 命令＋（st.[b|h|w] ／ sst.[b|h|w] ／ ld.[b|w] ／ sld.[b|w] 命令）＋分岐命令”の組み合わせに対し，最初の ld.w の直後に nop 命令を挿入する。

例

`ld.w`
`sst.w`
`jarl`
	`ld.w`
	`nop`
	`sst.w`
	`jarl`

(2) 2（-Xv850patch=2 → -p2）

“ld.w ／ sld.w ／ st.w ／ sst.w 命令＋分岐命令”の組み合わせに対し，ロード／ストア命令と分岐命令の間に nop 命令を挿入する。

例

`ld.w`
`jarl`
	`ld.w`
	`nop`
	`jarl`

なお，*num*=1 のパターンと同時に処理する場合，*num*=2 のパターンを先に検索して処理します。無駄な nop 命令を挿入することはありません。

(3) 3（-Xv850patch=3 → -p3）

reti 命令の直前に，対応する割り込みの制御レジスタに対し，clr1 命令を挿入する。

例

`reti`	`clr1  5, P0IC0` `reti`

(4) 4（-Xv850patch=4 → -p4）

“ロード命令（ld.[b|h|w] ／ sld.[b|h|w]）＋ロード・ストア命令（ld.[b|h|w] ／ sld.[b|h|w] ／ sst.[b|h|w] ／ st.[b|h|w]）”の組み合わせに対し，最初のロード命令の直後に nop 命令を挿入する（入力ファイル中で周辺 I/O レジスタをアクセスしている場合に挿入する）。

例

`ld.w`
`ld.w`
	`ld.w`
	`nop`
	`ld.w`

(5) 4a（-Xv850patch=4a → -p4a）

“ロード命令（ld.[b|h|w] ／ sld.[b|h|w]**）＋ロード・ストア命令（**ld.[b|h|w] ／ sld.[b|h|w] ／ sst.[b|h|w] ／ st.[b|h|w]**）”の組み合わせに対し，最初のロード命令の直後に** nop **命令を挿入する（周辺** I/O **レジスタ・アクセスの有無にかかわらず挿入する）。**

例

`ld.w`
`ld.w`
	`ld.w`
	`nop`
	`ld.w`

-p4 **は，入力ファイル中で周辺** I/O **レジスタ・アクセスがあった場合，**4 **のパッチを当てる。**

-p4a **は，周辺** I/O **レジスタ・アクセスの有無に関わらず** 4 **のパッチを当てる。**

(6) 5（-Xv850patch=5 → -p5）

乗算命令に対し，無条件で直後に nop **命令を挿入する。**

例

`mulh`
`jarl`
	`mulh`
	`nop`
	`jarl`

(7) 6（-Xv850patch=6 → -p6）

“ロード命令（ld.[b|h|w] ／ sld.[b|h|w]**）＋** jr ／ jarl ／ jcond（bcond）**”の組み合わせに対し，ロード命令の直後に** nop **命令を挿入する。**

例

`sld.bu`
`jarl`
	`sld.bu`
	`nop`
	`jarl`

(8) 7（-Xv850patch=7 → -p7）

callt **命令の直後に** nop **命令を挿入する。また，**switch **命令，および** reti **命令の直前に“** mov r31, r0 **”命令を挿入する。**

例

`switch`	`mov r31, r0` `switch`

(9) 8（-Xv850patch=8 → -p8）

連続する sld **命令の間に** nop **命令を挿入する。**

例

`sld.b`
`sld.b`
	`sld.b`
	`nop`
	`sld.b`

(10) 9（-Xv850patch=9 → -p9）

次の (a) ～ (c) に該当する命令が連続で存在した場合，(b) の sld 命令の直後に nop 命令を挿入する。

例

add sld and	add … (a) sld.b … (b) nop and … (c)

(a) 2 **バイト長の** mov，not，satsubr，satsub，satadd，zxb，zxh，sxh，or，xor，and，subr，sub，add，shr，sar，shl **命令のうち** r0, r30 **以外にライト・バックする命令**

例

```
add    0x1, r10
```

ただし，LABEL**，式，または定義が参照より後にある** .set **シンボルを記述し，上記命令に命令展開される可能性のある命令が含まれます。**

次の例ではチップ・バグ・パターンにはあたりませんが，パッチを当てる対象となります。

例

```
addi   SYM, r10, r10
.set   SYM, 0x123
```

(b) (a) **の命令がライト・バックするレジスタと異なるレジスタにライト・バックする** sld **命令**

例

```
sld.b  %LABEL, r11
```

(c) (a) **の命令がライト・バックするレジスタ値をロードする命令**

例

```
add    r11, r10
```

ただし，LABEL**，式，または定義が参照より後ろにある** .set **シンボルを記述し，**(a) **の命令でライト・バックしたレジスタ値をロードする命令を記述した場合が含まれます。**

例

```
       addi   LABEL2-LABEL1, r10, r12
LABEL1:
       --（中略）
LABEL2:
```

この例では，LABEL2 **と** LABEL1 **の相対値が** 16 **ビットで表現できる範囲を越えた場合，次のように命令展開されます。**

```
mov    LABEL2-LABEL1, r12
and    r10, r12
```

ここで，(b) **の命令の直後は** mov **命令となり，**r10 **の値はロードされません。つまり，チップ・バグ・パターンには当てはまりませんが，パッチを当てる対象となります。**

(11) 10（-Xv850patch=10 → -p10）

“ストア命令（sst.[b|h|w] ／ st.[b|h|w]**）＋** jcond（bcond）**” の組み合わせに対し，ストア命令の直後に** nop **命令を挿入する。**

例

`sst.b`
`br`
	`sst.b`
	`nop`
	`br`

(12) *num* 指定なし（-Xv850patch → -p）

デバイス・ファイルより判断し，1 ～ 3，5 ～ 10 **の組み合わせの各コードを出力する（前記参照）。**

なお，パッチを当てる必要のないオブジェクト生成時にこのオプションを指定した場合，パッチを当てません。生成オブジェクトと各オプションの対応は次のとおりです。

表 4 － 5　**生成オブジェクトと** -p **オプションの対応**

生成するオブジェクト	-p1	-p2	-p3	-p4	-p4a	-p5	-p6	-p7	-p8	-p9	-p10
V850 **デバイス固定**	○	○	○	○	○	×	×	×	×	×	×
V850Ex **デバイス固定**	×	×	×	×	×	○	○	○	○	○	○
V850E2 **デバイス固定**	×	×	×	×	×	○	○	○	○	○	○
V850 **コア共通**	○	○	○	○	○	○	○	○	○	○	○
V850Ex **コア共通**	×	×	×	×	×	○	○	○	○	○	○
V850E2 **コア共通**	×	×	×	×	×	○	○	○	○	○	○

備考　**○：パッチを当てる**
　　×：パッチを当てない

第 5 章　リンカ

この章では，リンカ（ld850）の概要，操作方法，リンク・マップ，注意事項について説明します。

5.1　動作の流れ

アプリケーション・プログラムは，一般的に複数のソース・ファイルに分けられてコーディングされます。C 言語で書かれたソースはコンパイラ（ca850）／アセンブラ（as850）を，アセンブリ言語で書かれたソースはアセンブラ（as850）を起動し，オブジェクト・ファイルを出力します。

これらのオブジェクト・ファイル群を，リンク・ディレクティブ，およびデバイス・ファイルの情報に従ってアドレス解決し，実行可能な 1 つのオブジェクト・ファイル，つまり，ロード・モジュール・ファイルを生成するのが“ リンカ（ld850）”です。

リンカがオブジェクト・ファイル群をリンクするとき，未解決な外部参照があった場合は，指定されたアーカイブ・ファイル（ライブラリ・ファイル）を検索して解決しようとします。そして解決に必要なオブジェクト・ファイルのみをリンクし，実行可能なオブジェクト・ファイルを生成します。また，-r オプションによってリロケータブルなオブジェクト・ファイルを生成することができます。

図 5 － 1　ld850 における動作の流れ

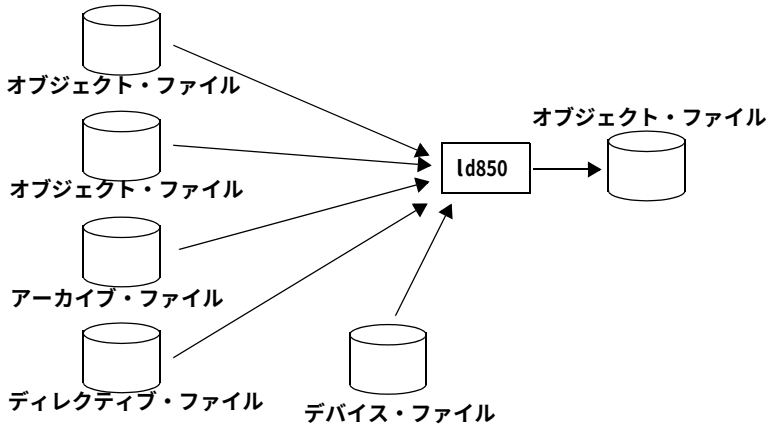
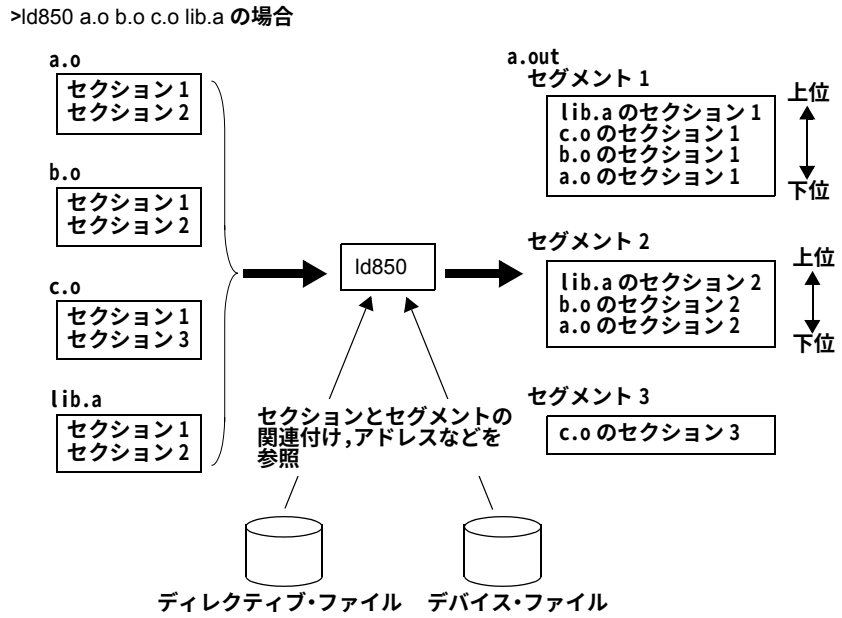


図 5 － 2 ld850 における動作のイメージ例



なお，ca850 はドライバとして as850，ld850 を内部で起動します。

ca850 を起動すると，ロード・モジュールまで生成することができます。つまり，as850 や ld850 の起動を意識する必要がありません。

図 5 － 3 一括処理

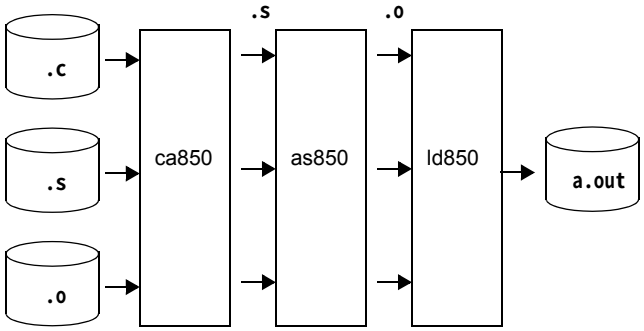
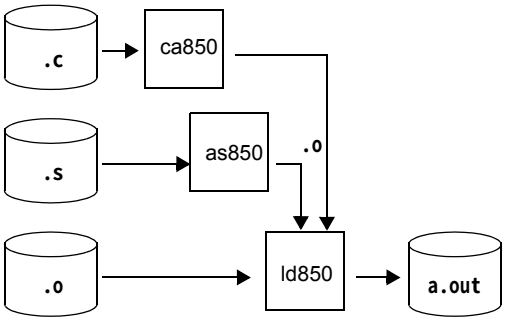


図 5 － 4 分割処理

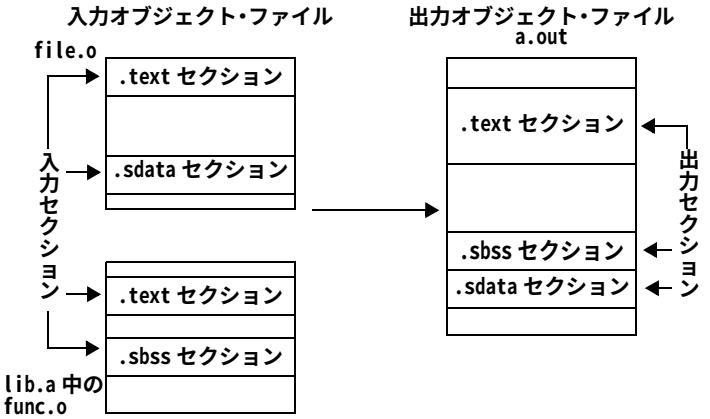


5.1.1 リンクの手順

ld850 におけるリンクの手順を次に示します。

(1) 指定されたオブジェクト・ファイルに含まれるセクション（入力セクション）をリンク・ディレクティブ，およびデバイス・ファイルに従って結合し，生成されるオブジェクト・ファイルを構成する出力セクションを作成する（“CA850 ユーザーズ・マニュアル リンク・ディレクティブ編 ” 参照）。

図 5 － 5 出力セクションの作成

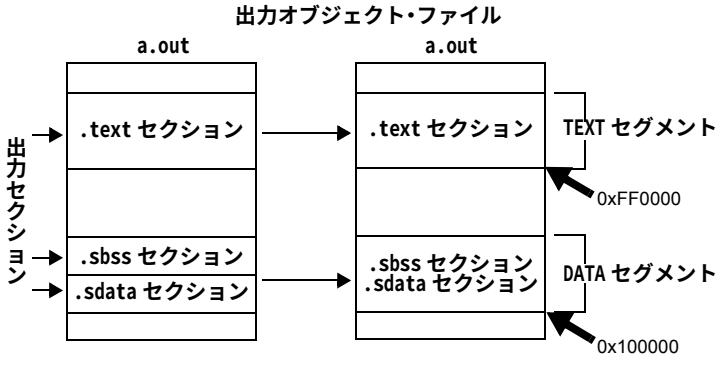


(2) (1) のステップにおいて作成された出力セクションをリンク・ディレクティブに従って結合し，セグメント[注]を作成する。

注 プログラムをメモリにロードする際の最小単位で，生成されるオブジェクト・ファイルのプログラム・ヘッダに反映されます。

(3) (2) のステップにおいて作成されたセグメントをリンク・ディレクティブ，およびデバイス・ファイルに従ってターゲット・マシンのメモリ空間に割り付ける。

図 5 － 6 メモリ空間への割り付け



(4) 出力セクション内の未解決な外部参照を解決する。

(5) リンク・ディレクティブ内のシンボル・ディレクティブに従い，次の 3 種類のシンボルを生成する[注]。

- テキスト・ポインタ（tp）に設定する値を持つテキスト・ポインタ・シンボル
- グローバル・ポインタ（gp）に設定する値を持つグローバル・ポインタ・シンボル
- エレメント・ポインタ（ep）に設定する値を持つエレメント・ポインタ・シンボル

注 これらのシンボルは（たとえば，スタート・アップ・モジュールにおいて），CA850 によって生成されたコードを実行する前にテキスト・ポインタ（tp），グローバル・ポインタ（gp），およびエレメント・ポインタ（ep）を適切な値に設定する場合に用いることができます。

エレメント・ポインタ値はユーザが設定することもできますが，省略した場合，エレメント・ポインタ・シンボルには，ld850 がターゲット・デバイス固有の値（内蔵 RAM の先頭アドレス）を，指定したデバイス・ファイルより読み込み，設定します。

(6) 予約シンボルを生成する。予約シンボルには次のものがあります。

- 各出力セクションの先頭アドレス
- 各出力セクションの終端を越える最初の（4 バイトで整列された）アドレス
- 生成された実行可能なオブジェクト・ファイルの終端を越える最初の（4 バイトで整列された）アドレス

予約シンボルの詳細は「5.7.3 予約シンボル」を参照してください。

5.2 操作方法

この節では，ld850 の操作方法について説明します。

5.2.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
ld850 [ オプション ]... ファイル名 [ ファイル名，またはオプション ]...
       [ ]    ：[ ] 内は省略できます。
        ...   ：直前の [ ] 内のパターンの繰り返しができます。
```

5.2.2 PM+ による方法

[リンカオプションの設定] ダイアログは，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [リンカオプションの設定] を選択

リンカの起動は，プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

リンカが出力する実行可能オブジェクト・ファイル名は，a.out となります。
なお，ファイル名は，[ファイル] タブの “ 出力ファイル [-o] ” で指定することもできます。

5.3 オプションの種類と機能

この節では，ld850 のオプションを示します。

なお，ld850 のオプションには，PM+ 上では設定できないオプションが存在します。このようなオプション指定が必要な場合は，コマンド・ラインから ld850 を起動してください。

〔オプション説明でのマーク〕

【V850E2】	V850E2 コア専用のオプション
【V850E】	V850Ex コア専用のオプション
【PM+】	PM+ で指定項目が存在するオプション

5.3.1 入力ファイル

ld850 の入力ファイルに関するオプションの設定を行います。

-D *dfile*

【PM+】

リンク・ディレクティブ・ファイル *dfile* 内のリンク・ディレクティブに従ってリンクします。*dfile* の長さは，パスを指定する部分を含んだ長さで 127 文字以内，パスを指定する部分を含んでいない長さで 14 文字以内にしてください。

なお，拡張子も必要です。推奨する拡張子は “ .dir ” です。このオプションを省略した場合，デフォルトのリンク・ディレクティブを用います。

リンク・ディレクティブ・ファイルについての詳細は “ CA850 ユーザーズ・マニュアル リンク・ディレクティブ編 ” を参照してください。

-Xolddir[=*version*]

【PM+】

リンク・ディレクティブのフォーマットの，旧版との互換性を選択します。version には “ V240 ”, “ V250 ”, “ V260 ” を指定できます。version 省略時には “ V240 ” が指定されたものとして扱います。

このオプションを指定しなかった場合は，最新のリンク・ディレクティブのフォーマットに対応します。

- V240 指定時

  セクション優先配置機能 OFF，セグメント・ソート OFF（CA850 Ver.2.40 相当）
- V250 指定時

  セクション優先配置機能 ON，セグメント・ソート OFF（CA850 Ver.2.50 相当）
- V260 指定時

  セクション優先配置機能 ON，セグメント・ソート ON（CA850 Ver.2.60 以降相当）

5.3.2 出力ファイル

ld850 が参照，リンクするファイルと出力ファイル名を指定します。

+err_file=*file*

エラー・メッセージをファイル *file* に追加保存します。

-err_file=*file*

エラー・メッセージをファイル *file* に上書き保存します。

-o *ofile*

【PM+】

生成するオブジェクト・ファイル名を *ofile* とします。このオプションを省略した場合，a.out が指定されたものとみなします。

-m[=*mapfile*]

【PM+】

入力セクション，出力セクションのメモリ空間への割り付け状態を示すリンク・マップを *mapfile* に出力します。*mapfile* を省略した場合，標準出力に出力します。

リンク・マップに関しての詳細は「5.5 リンク・マップ」を参照してください。

-mo[=*mapfile*]

【PM+】

入力セクション，出力セクションのメモリ空間への割り付け状態を示すリンク・マップを CA850 Ver.2.60 以前の旧形式で *mapfile* に出力します。*mapfile* を省略した場合，標準出力に出力します。

5.3.3 ライブラリ

ld850 が参照するライブラリに関するオプションを指定します。

-L*dir*

【PM+】

このオプションとともに（コマンド・ラインからの場合このオプションのあとに）-l オプションが指定された場合，-l オプションによって指定されたアーカイブ・ファイル（ライブラリ・ファイル）を，フォルダ dir，標準フォルダの順で探します。このオプション以降に指定した -l オプションが対象となります。

このオプションを省略した場合，標準フォルダで探します。

ld850 は，ld850 のインストールされたフォルダから ..\lib850 の位置にあるフォルダ，および ..\lib850\r*XY* の位置にあるフォルダ（*XY*=[32 | 26 | 22]）をライブラリに対する標準フォルダとして扱います。

-lc

【PM+】

コンパイラの標準ライブラリ（libc.a）をリンクします。

-lm

【PM+】

コンパイラの数学ライブラリ（libm.a）をリンクします。

なお，数学ライブラリは，標準ライブラリ内の関数も参照するため，-lc オプションが設定されている場合に有効です。

-l*string*

【PM+】

未解決な外部シンボルの参照を解決する際，アーカイブ・ファイル lib*string*.a を参照します。このオプションで複数のアーカイブ・ファイルが指定された場合，指定順序に従って検索します。

(1) *string* の文字数は，64 文字未満にしてください。

(2) ld850 は，このオプションの指定において，指定された時点で未解決な外部参照についてのみ指定されたアーカイブ・ファイルを参照します。したがって，コマンド・ラインからの起動では，このオプションは，指定するアーカイブ・ファイルを参照するオブジェクト・ファイルより，後ろに指定してください。

(3) CA850 で提供している数学ライブラリは，標準ライブラリの libc.a ファイルを参照しています。したがって，コマンド・ラインからの起動では，標準ライブラリの参照指定 “ -lc ” は，数学ライブラリの参照指定 “ -lm ” より後ろに指定してください。

5.3.4 フラッシュ

ld850 のフラッシュに関するオプションを指定します。

`-ext_table` ***address***

【PM+】

8 桁の 16 進数 *address* で指定される値を分岐テーブル先頭アドレス値として，フラッシュ／外付け ROM 再リンク機能用のオブジェクト・ファイルを生成します（「5.6 フラッシュ／外付け ROM 再リンク機能」を参照）。

- ブート領域側指定時には，フラッシュ領域側への分岐処理となります。
  この際，分岐テーブルへの分岐となりますが，このアドレスが，このオプションで指定されるアドレスとなります。
- フラッシュ領域側指定時には，このオプションに指定したアドレスに，本来の分岐先への分岐命令を持つ分岐テーブルを生成します。
- このオプションに指定するアドレス値は，ブート領域側／フラッシュ領域側作成の際に同じ値とする必要があります。異なる値を指定した場合，正しく動作しません。また，エラー・チェックもしていません。
- このオプションに指定するアドレス値は，フラッシュ領域側 ROM 内である必要があります。指定したアドレスがどちらの領域であるかの判断ができないため，エラー・チェックはしていません。
- このオプションにより，フラッシュ領域側作成時には，指定されたアドレス値を先頭とする，サイズ（(ID 値[注]の最大 +1）×分岐テーブルのエントリサイズ）バイトのセクション .ext_table を自動作成します。このセクションは，ディレクティブ・ファイルで配置指定を行う必要はありませんが，配置するため領域を空けておく必要があります。
- このオプションは，-r オプションとの併用はできません。また，-r オプションにより生成したリロケータブル・オブジェクト・ファイルを入力した場合，正しく動作しません。
  フラッシュ／外付け ROM 再リンク機能の詳細は「5.6 フラッシュ／外付け ROM 再リンク機能」を参照してください。

注 アセンブリ言語ソース・ファイルに .ext_func 疑似命令で指定された値

このオプションを省略した場合，フラッシュ／外付け ROM 再リンク機能用のオブジェクト・ファイルを生成しません。

-zf *bootfile*

【PM+】

フラッシュ／外付け ROM **再リンク機能使用時に，指定したオブジェクト・ファイルをブート領域側のオブジェクト・ファイルとして，フラッシュ側オブジェクト・ファイルを生成します。**

ブート領域側オブジェクト・ファイルは，フラッシュ／外付け ROM **再リンク機能を指定して作成したものを指定してください。**

なお，ここで指定するものは，ld850 **が出力したオブジェクトです。**romp850 **が出力したオブジェクトを指定すると，不正なオブジェクトが生成されるので注意してください。このオプションを省略した場合，ブート領域側オブジェクト・ファイルを生成します。このオプションを使用する場合には，**-ext_table **オプションが指定されている必要があります。**

例

```
ld850 -ext_table 0x200 -D dfile2 -zf boot.out -o flash.out flash1.o flash2.o
```

(1) **フラッシュ／外付け** ROM **再リンク機能用に処理されたアセンブリ言語ソース・ファイルから作成したオブジェクト・ファイル** flash1.o **と** flash2.o **をフラッシュ領域オブジェクトとしてリンクします。**

(2) 0x200 **番地に分岐テーブルを生成します。**

(3) **リンク時に，ブート領域実行可能形式オブジェクト・ファイル** *boot.out* **の情報を参照します。この際，** *boot.out* **はリンクされません。**

(4) **フラッシュ領域側実行可能形式ディレクティブ・ファイルに従ってリンクします。**

(5) **生成されるオブジェクト・ファイル名を** *flash.out* **とします。**

5.3.5 デバイス

ld850 のデバイスに関するオプションを指定します。

-X256M

【V850E】【PM+】

メモリ空間を 256M バイトとして扱います。このオプションを指定しない場合はメモリ空間を 64M バイトとして扱い，アドレス解決します。このオプションは使用するチップセットにあわせて設定してください。

V850Ex コアでは，物理アドレス空間が 256M バイト持つ場合が多く，64M を越えた 256M までの空間を使用するアプリケーションを作成する場合，このオプションを指定してください。

-Xsid=*id*

【PM+】

フラッシュ・メモリ搭載デバイスの“ セキュリティ ID ”を設定します。セキュリティ ID 機能をサポートしていないデバイスの場合は利用できません。ID は，10 バイト以内の 16 進数（先頭の 0x も含む）で指定します。

なお，指定した値が 10 バイトより不足している場合は，上位ビットを 0 で埋めます。10 バイトを越えた場合はエラーを出力します。

セキュリティ ID 機能をサポートしているデバイスに対して，本オプションの指定，またはアセンブラ記述 (.section "SECURITY_ID" 使用) によるセキュリティ ID の指定が省略された場合，“ 0xffffffffffffffffffff ”が指定されたものとして扱います。

上記以外の方法でセキュリティ ID を設定した場合，リンカが生成したセキュリティ ID により多重指定したものと判断し，次のエラーとなります。

F4264: start address(0x00000070) of section "SECURITY_ID" overlaps previous section " 任意のセクション名 " ended before address (0xXXXXXXXX).

このような場合には，+Xsid オプションを指定し，リンカによるセキュリティ ID の生成を抑止してください。セキュリティ ID 機能をサポートしていないデバイス用のオブジェクトをリンク時に指定した場合は，警告を出力し，指定は無視されます。

たとえば，セキュリティ・コード 0x112233445566778899aa を（0x70 番地に 0x11，0x71 番地に 0x22，0x72 番地に 0x33，0x73 番地に 0x44，0x74 番地に 0x55，0x76 番地に 0x77，0x77 番地に 0x88，0x78 番地に 0x99，0x79 番地に 0xaa）設定する場合は，次のように記述します。

```
-Xsid=0x112233445566778899aa
```

-Xob=none

デフォルトで生成されるオプションバイトを抑止しします。

本オプションは，デバイス・ファイルに登録された初期値によるデフォルト生成のみを抑止します。アセンブラ・ソース・ファイルで .section "OPTION_BYTES" を使用して指定が行われた場合には，本オプションの指定の有無にかかわらず，.section "OPTION_BYTES" の指定が優先されます。

なお，本オプションをオプション・バイト機能を持たないデバイスに対して指定した場合には，メッセージを出力せずに本オプションを無視します。

5.3.6 オプション

ld850 のオプションを指定します。

-A

【PM+】

ソース・ファイルのコンパイル時，およびアセンブル時に ca850，および as850 に対して指定する [sdata ／ sbss のデータ配置] オプション（-G*num* オプションの *num* の設定）において，目安として用いることのできる情報を標準出力に出力します。*OK* と表示された数値を用いると，sdata ／ sbss の領域へは，その数値以下のサイズを持つデータが割り当てられます。

なお，ca850 から起動された場合，ca850 の起動時に指定された -A が渡されます。

詳しくは，「5.7.1 -A オプションの使い方」を参照してください。

-B

【PM+】

2 パス・モードでリンクを行います。このオプションを省略した場合，1 パス・モードでリンクを行います。2 パス・モードは 1 パス・モードよりも低速ですが，より大きなサイズのファイルを処理できます。

-E

【PM+】

リロケーション処理において，次の不正箇所があった場合，

- 未解決な外部参照のアドレス計算結果が不正な場合
- 配置されるセクションとの関係が不正な場合

エラーとはせず，警告メッセージを出力してリンクを続行します。

誤りとされた未解決な外部参照に対しては，不正と判断されたアドレスの計算結果の値は入れられず，元の値が残されます。

-M

【PM+】

多重定義されたすべての外部シンボルに対してメッセージを出力し，リンクの処理を中止します。このオプションを省略した場合，多重定義された最初の外部シンボルに対してメッセージを出力し，リンクの処理を中止します。

-T

【PM+】

外部シンボルのリンクの際，サイズ，および整列条件のチェックを行いません。このオプションを省略した場合，サイズのチェックを行い，サイズの違いが検出された場合，警告メッセージを出力し，リンクを続行します。この際，実際にシンボルが定義されているファイルのシンボル・サイズが有効となります。

-Ximem_overflow=warning

【PM+】

内蔵 ROM ／ RAM をオーバーフローした際のチェックの制御を行います。オーバーフロー時には警告メッセージを出力し，リンクを継続します。

-e *symbol*

【PM+】

シンボル *symbol* 値を生成されるオブジェクト・ファイルのエントリ・ポイント・アドレス値とします。指定したシンボルが見つからない場合，リンカはメッセージを出力してリンクを中止します。このオプションを指定しない場合，次の規則でエントリ・ポイント・アドレス値を定めます。

(1) シンボル __start が存在する場合は，その値

(2) __start が存在しない場合は，生成されるオブジェクト・ファイル内の最下位に割り付けられた text 属性のセクションの先頭アドレス

(3) text 属性セクションが存在しない場合は 0

なお，シンボル名に空白は使用できません。

-f *num*

【PM+】

生成されるオブジェクト内のセクション間のアライン・ホールのフィリング値を，4 桁の 16 進数（2 バイト分）で指定します。このオプションを用いる場合は，-B オプションを指定して 2 パス・モードでリンクを行ってください。先頭の “ 0x ” は省略可能です。このオプションによる指定は，リンク・ディレクティブにおけるフィリング値指定より優先します。

- 4 桁に満たない場合，満たない分の 0 が頭に指定されたものとみなします。
- ホールの大きさが 2 バイトに満たない場合，指定されたフィリング値の下位から必要な桁数分だけを取り出して初期化を行います。

このオプションを省略した場合，0x0000 が指定されたものとみなします。

-mc

【PM+】

C 言語ソース・ファイルから作成されたオブジェクト・ファイルのリンクの際，マスク・レジスタ機能を使用しているファイルと，使用していないファイルが混在していないかどうかチェックします。混在している場合，リンクを中止します。

-rc

【PM+】

すべての入力オブジェクト・ファイルに対し，異なるレジスタ・モードが混在している場合に詳細な情報を出力します。-w オプションと同時指定された場合，このオプションは無視されます。このオプションを省略した場合，詳細な情報を出力しません。

-rescan

【PM+】

-l オプションで指定したライブラリ・ファイルの再参照を行います。このオプションを指定すると，ライブラリのリンク順によるシンボル未解決を防ぐことができます。

-rom_less

【PM+】

内蔵 ROM 領域に対する配置に対して，チェックを行いません。つまり，内蔵 ROM 領域となっているアドレスに，アプリケーションの配置がオーバラップしていても，警告メッセージを出力しません。アプリケーションを ROM レス・モードで作成する場合，このオプションを指定してください。

注意 シングルチップ・モード選択時の内蔵 ROM オーバのチェックには対応していません。このオプションを指定して内蔵 ROM オーバーフローのチェックを無効とし，リンク・マップで確認してください。

-s

【PM+】

オブジェクト・ファイルの生成において，デバッグ情報，ライン・ナンバ情報，およびグローバル・ポインタ・テーブルを取ったオブジェクト・ファイルを生成します。

-t

【PM+】

未定義外部シンボルのリンクにおいて，シンボルのサイズ，および整列条件のチェックを行いません。このオプションを省略した場合，シンボル・サイズ，および整列条件のチェックを行い，違いが検出された場合，警告メッセージを出力し，リンクを続行します。

- リンカは，未定義外部シンボルの多重定義をサポートしています。多重定義された未定義外部シンボルは，リンク後 .sbss，または .bss セクションに割り付けられます。この際，リンクされるシンボル・サイズ，または整列条件が異なっていた場合，サイズは，リンクされるシンボルのうちの最大サイズとし，整列条件は，リンクされるシンボルの整列条件の最小公倍数とします。

-v

【PM+】

ld850 の実行状況の詳細を出力します。リンクするオブジェクトの一覧等が表示されます。

-w

【PM+】

警告メッセージを出力しません。致命的な誤りに対するメッセージのみを出力します。

5.3.7 その他

その他のオプションを設定します。

-F *devpath*

ld850 を単体で起動する場合，デバイス・ファイルを，フォルダ *devpath* から探します。このオプションを省略した場合，標準フォルダから探します。ca850 から起動する場合に，デバイス・ファイルのパスを指定するには，ca850 の -devpath オプションを使用します。

-V

ld850 のバージョン情報を標準エラー出力に出力し，終了します。

-cpu *devicename*

devicename で指定されたターゲット・デバイスのデバイス・ファイルを読み込みます。PM+ 使用の場合，プロジェクトの設定時に行うデバイス指定に相当します。このオプションを省略した場合，.o ファイル作成時に指定されたターゲット・デバイスのデバイス・ファイルを読み込みます。

-fc

すべての入力オブジェクト・ファイルに対し，旧関数呼び出しと現バージョンの呼び出し仕様が混在していないかチェックします。旧関数呼び出し仕様は現バージョンではサポート対象外です。このオプションを省略した場合，C 言語ソース・ファイルから作成されたオブジェクト・ファイルのみをチェックします。

-help

オプションの説明を標準エラー出力に出力します。

-mask_reg

マスク・レジスタ機能用のライブラリを参照します。ca850 から起動する場合，-Xmask_reg オプションを使用します。マスク・レジスタ機能用のライブラリは，32 レジスタ・モード時に使用できるライブラリです。22，26 レジスタ・モード時に指定した場合，次の警告メッセージを出力し，後ろに指定したものを無視します。

```
W4857: reg22" option is illegal when "-mask_reg" option is specified, ignored
"-reg22" option.
```

-r

リロケータブル（再配置可能）なオブジェクト・ファイルを生成します。-ro オプションと同時に指定された場合，このオプションは無視されます。このオプションを指定すると，リンク終了後に未解決な外部参照が残されていてもメッセージを出力せずにリンクを正常終了します。このオプションを省略した場合，未解決な外部参照が残されていた場合は，次のメッセージを出力し，リンクを中止します。この場合，オブジェクト・ファイル（ロード・モジュール・ファイル）は生成されません。

```
F4452：undefined symbol.
       symbol referenced in "file"
```

また，ld850 によって生成されたオブジェクト・ファイルを，ld850 による再リンクの対象として指定する場合，再リンクの対象となるオブジェクト・ファイルの生成には，このオプションを使用してください。

[注意事項]

- このオプションが指定されると，リンク・ディレクティブは，マッピング・ディレクティブの部分のタイプ／属性のみが有効になり，その他は無視されます。
- このオプションが指定されると，予約シンボルの作成を行いません。
- CA850 Ver.2.30 以前からは，-r オプションの仕様が変更されています。
  旧バージョンのマッピング方式を用いる場合，-ro オプションの代わりに -r を用いてください。

-ro

リロケータブル（再配置可能）なオブジェクト・ファイルを旧マッピング方式（CA850 Ver.2.30 以前）で生成します。-r オプションと同時指定された場合は， -r は無視されます。このオプションを省略した場合，実行可能なオブジェクト・ファイルを生成します。

-reg*num*

対応するレジスタ・モード用のライブラリを参照します。*num* には，22，26，または 32 が指定可能です。-reg の後ろに空白を入れないでください。このオプションを省略した場合，32 レジスタ・モード用のライブラリを参照します。

@cfile

cfile をコマンド・ファイルとして扱います。Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。このオプションを指定している場合は，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，文字列の制限を意識する必要がなくなります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

5.4 PM+ での設定

この節では，対象プロジェクトのソース・ファイルに対して，ld850 のコマンド・オプションを設定するダイアログについて説明します。

5.4.1 [リンカオプションの設定] ダイアログ

リンカオプションの設定ダイアログの上部には，次の 4 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

表 5 － 1 [リンカオプションの設定] ダイアログ

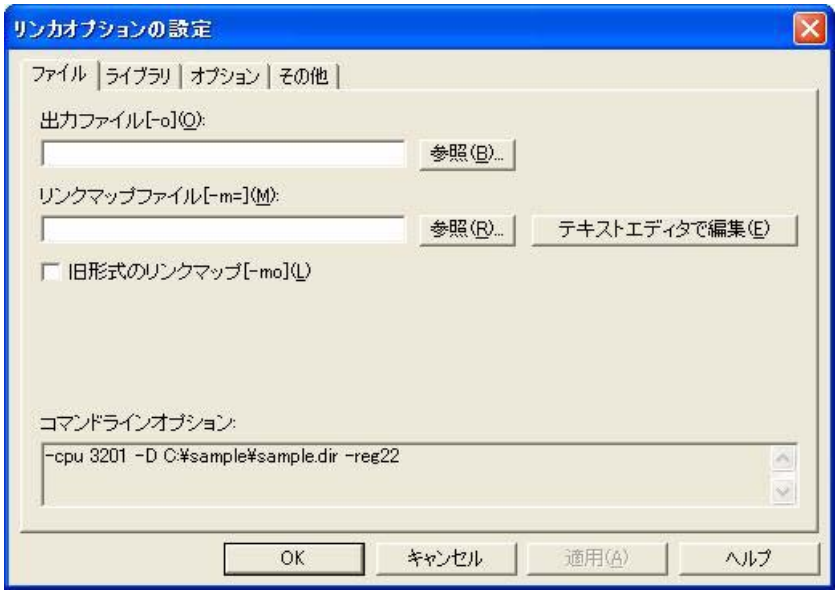
タブ	内容
[ファイル]	ファイルに関する設定
[ライブラリ]	ライブラリに関する設定
[オプション]	リンカのオプションに関する設定
[その他]	その他の設定

なお，ダイアログ上の “ [] ” 内に表示されたオプション名は，コマンド・プロンプトから起動する場合のオプション名です。

[ファイル]

リンカのファイルに関する設定を行います。

図 5 － 7 [リンカオプションの設定] ダイアログ：[ファイル] タブ



(1) 出力ファイル [-o]

出力するオブジェクト・ファイル名（拡張子は“ .out ”）を指定します。このオプションを省略した場合，a.out が指定されたものとみなします。また，[参照] ボタンにより表示されるダイアログで，ファイルを選択することができます。

(2) リンクマップファイル [-m=]

リンク・マップを出力する際，出力先のファイル名を指定します。また，[参照] ボタンにより表示されるダイアログで，ファイルを選択することができ，[編集] ボタンにより，テキスト・エディタを表示し，リンク・マップ・ファイルを編集することができます。

なお，この指定は [オプション] タブの “ リンクマップの表示 [-m] ” より優先されます。

(3) 旧形式のリンクマップ [-mo]

旧形式のリンクマップを出力するかどうかを設定します。チェックされている場合は，“ リンクマップファイル [-m=] ” で指定されたリンク・マップ・ファイルは，旧形式で出力されます。

(4) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[ライブラリ]

リンカのライブラリに関する設定を行います。

図 5 － 8 [リンカオプションの設定] ダイアログ：[ライブラリ] タブ

(1) ライブラリ [-l]

参照するアーカイブ・ファイル（ライブラリ・ファイル）lib*string*.a の *string* 部分を指定します。複数指定する場合，“ ; （セミコロン）” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，ライブラリ項目をダイアログ上で編集することができます。

アーカイブ・ファイルは，“ ライブラリのパス ” で指定したフォルダ，ライブラリの標準フォルダの順に検索します。

リンカは，指定したファイルが見つからなかった場合，何のメッセージも出力せずにリンクを続行します。

(2) ライブラリのパス [-L]

参照するアーカイブ・ファイル（ライブラリ・ファイル）のフォルダを指定します。複数指定する場合，“ ; （セミコロン）” で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，パス項目をダイアログ上で編集することができます。このオプションでフォルダを指定した場合，ライブラリの標準フォルダより先に指定フォルダでライブラリを検索します。複数指定した場合，テキスト・ボックスに指定した順に検索します。

(3) 標準ライブラリをリンク [-lc]

パッケージで提供している標準ライブラリ “ libc.a ” を参照します。このオプションはデフォルトで選択されています。

(4) 数学ライブラリをリンク [-lm]

数学ライブラリは標準ライブラリを参照しているため，参照ライブラリとして標準ライブラリもチェックしてください。

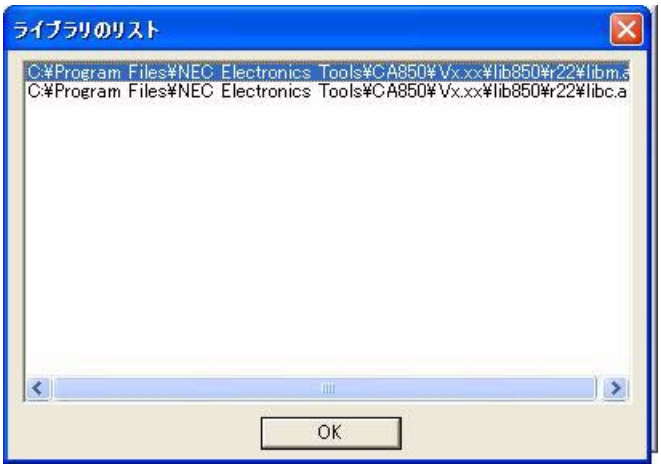
なお，ライブラリ参照は自動的に入力オブジェクト・ファイルの後ろに指定され，標準ライブラリは，他のライブラリより後ろにリンクされます。また，ライブラリのパス指定は，自動的にライブラリ参照の前に指定されます。

また，ROM 化する場合に必要な rompcrt.o は，自動的にライブラリより後ろにリンクされます。これは ROM 化用の rompcrt セクションを，他の入力ファイルの .text セクションより後ろにするためです。

(5) ライブラリのリスト

このボタンを選択すると，現在このページで設定されているオプションによってリンク対象になるライブラリのリストを表示します。

図 5 － 9 ライブラリのリスト



(6) コマンドラインオプション

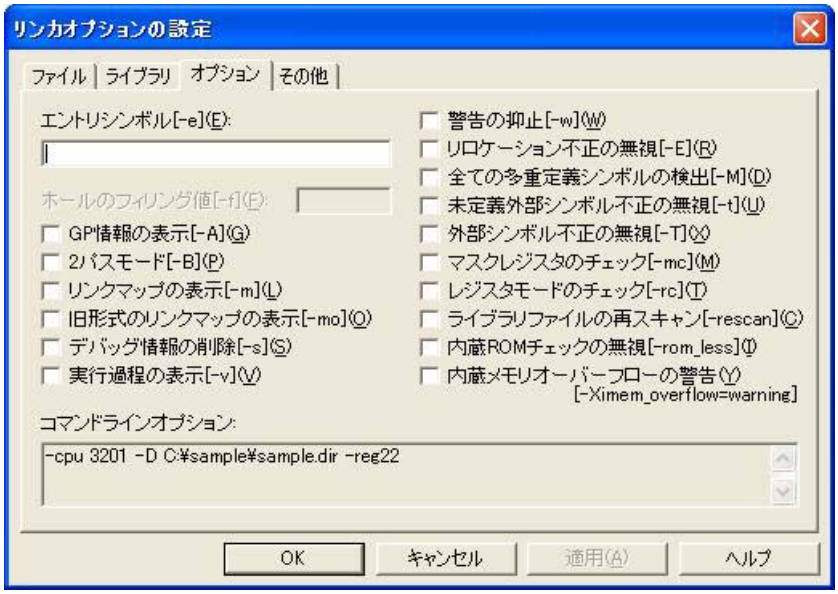
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[オプション]

リンカのオプションに関する設定を行います。

図 5 － 10 [リンカオプションの設定] ダイアログ：[オプション] タブ



(1) エントリシンボル [-e]

エントリ・ポイント・アドレスとして設定するシンボルを指定します。指定したシンボルが見つからない場合，リンカはメッセージを出力してリンクを中止します。

このオプションを指定しない場合，次の規則でエントリ・ポイント・アドレス値を定めます。

- シンボル __start（アンダスコア ‘ _ ’ 2 つ）が存在する場合，その値
- __start が存在しない場合，生成されるオブジェクト・ファイル内の最下位に割り付けられた text 属性セクションの先頭アドレス
- text 属性セクションが存在しない場合，0

(2) ホールのフィリング値 [-f]

生成されるオブジェクト内のホールのフィリング値を，4 桁の 16 進数（2 バイト分）で指定します。このオプションを用いる場合は，“ 2 パスモード [-B] ” を指定してリンクを行ってください。先頭の “ 0x ” は省略可能です。このオプションによる指定は，リンク・ディレクティブにおけるフィリング値指定より優先されます。4 桁に満たない場合，満たない分の 0 が頭に指定されます。ホールの大きさが 2 バイトに満たない場合，指定されたフィリング値の下位から必要な桁数分だけ取り出します。省略した場合，0x0 となります。

(3) GP 情報の表示 [-A]

ソース・ファイルのコンパイル時，およびアセンブル時に，ca850 の出力コードや as850 のオプションで指定する“ sdata ／ sbss セクションに配置するデータ長の上限オプション ”の数値設定において，目安として用いることのできる情報を PM+ のアウトプット・ウインドウに出力します。*OK* と表示された数値を用いると sdata ／ sbss の領域へは，その数値以下のサイズを持つデータが割り当てられます。

なお，表示結果は，プロジェクト・フォルダに自動生成されるログ・ファイル（プロジェクト名＋ .plg）に含まれます。

sdata ／ sbss の割り当てデータ・サイズの制限についての詳細は，[出力コード] タブの“ sdata ／ sbss セクションに配置するデータ長の上限 [-G] ”を参照してください。

(4) 2 パスモード [-B]

-B オプションで指定する 2 パス・モードでリンクするかどうかを設定するチェック・ボックスです。

(5) リンクマップの表示 [-m]

入力セクション，出力セクションのメモリ空間への割り付け状態を示すリンク・マップを，PM+ のアウトプット・ウインドウに出力します。

なお，出力されたリンク・マップは，プロジェクト・フォルダに自動生成されるログ・ファイル（プロジェクト名＋ .plg）に含まれます。

備考 [ファイル] タブの“ リンクマップファイル [-m=] ”で出力先ファイル名を指定した場合，ファイルへの出力が優先され，このオプションは無効となります。

(6) 旧形式のリンクマップの表示 [-mo]

旧形式のリンクマップを出力するかどうかを設定します。[ファイル] タブの“ 旧形式のリンクマップ [-mo] ”と同じです。

(7) デバッグ情報の削除 [-s]

デバッグ情報，ライン・ナンバ情報，およびグローバル・ポインタ・テーブルを除いたオブジェクト・ファイルを生成します。このオプションはビルドモードが“ Release Build ”の場合にのみ指定可能です。

(8) 実行過程の表示 [-v]

ld850 の実行状況の詳細をアウトプット・ウインドウに出力します。リンクするオブジェクトの一覧等が表示されます。

(9) 警告の抑止 [-w]

警告メッセージを出力しません。致命的な誤りに対するメッセージのみを出力します。

(10) リロケーション不正の無視 [-E]

リロケーション処理において，次の不正箇所があった場合，

- 未解決な外部参照のアドレス計算結果が不正な場合
- 配置されるセクションとの関係が不正な場合

エラーとはせず，警告メッセージを出力してリンクを続行します。

誤りとされた未解決な外部参照に対しては，不正と判断されたアドレスの計算結果の値は入れられず，元の値が残されます。

(11) 全ての多重定義シンボルの検出 [-M]

すべての多重定義された外部シンボルに対してメッセージを出力し，リンクを中止します。このオプションを指定しなかった場合，多重定義された最初の外部シンボルに対してのみメッセージを出力し，リンクを中止します。

(12) 未定義外部シンボル不正の無視 [-t]

未定義外部シンボル・リンク時のサイズ，および整列条件のチェックを無視します。このオプションを指定しなかった場合，シンボル・サイズ，および整列条件のチェックを行い，違いが検出された場合，警告メッセージを出力し，リンクを続行します。

備考 リンカは，未定義外部シンボルの多重定義をサポートしています。多重定義された未定義外部シンボルは，リンク後 .sbss，または .bss セクションに割り付けられます。この際，リンクされるシンボル・サイズ，または整列条件が異なっていた場合，サイズは，リンクされるシンボルのうちの最大サイズとし，整列条件は，リンクされるシンボルの整列条件の最小公倍数とします。

(13) 外部シンボル不正の無視 [-T]

外部シンボルのリンクの際，サイズのチェックを行いません。このオプションを指定しなかった場合，サイズのチェックを行い，サイズの違いが検出された場合，警告メッセージを出力し，リンクを続行します。この際，実際にシンボルが定義されているファイルのシンボル・サイズが有効となります。

(14) マスクレジスタのチェック [-mc]

C 言語ソース・ファイルから作成されたオブジェクト・ファイルのリンクの際，マスク・レジスタ機能を使用しているファイルと使用していないファイルが混在していないかどうかチェックします。混在している場合，リンクを中止します。

(15) レジスタモードのチェック [-rc]

すべての入力オブジェクト・ファイルに対し，異なるレジスタ・モードが混在していないかどうかチェックし，混在する場合，情報を出力します。

なお，“ 警告の抑止 ” をチェックした場合，選択できません。

(16) ライブラリファイルの再スキャン [-rescan]

[ライブラリ] タブの“ ライブラリ [-l] ”で指定したライブラリ・ファイルを再スキャンするかどうかを指定します。このオプションを指定すると，ライブラリのリンク順によるシンボル未解決を防ぐことができます。

(17) 内蔵 ROM チェックの無視 [-rom_less]

内蔵 ROM 領域に対する配置に対して，チェックを行いません。内蔵 ROM 領域となっているアドレスに，アプリケーションの配置がオーバラップしていても，警告メッセージを出力しません。このオプションは，ROM レス・モード使用時に指定してください。不要な警告メッセージが出力されなくなります。

(18) 内蔵メモリオーバーフローの警告 [-Ximem_overflow=warning]

内蔵 ROM ／ RAM をオーバーフローした際のチェックの制御を行います。オーバーフロー時には警告メッセージを出力し，リンクを継続します。

(19) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[その他]

リンカのその他の設定を行います。

図 5 － 11 [リンカオプションの設定] ダイアログ：[その他] タブ

(1) 他のオプション

前記の “ リンカオプションの設定 ” では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

なお，リンカに関するオプションは，[リンカオプションの設定] ダイアログですべて指定できるため，他のオプションを使用する必要はありません。

(2) コマンドファイルの作成

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。チェックされている場合は，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，文字列の制限を意識する必要がなくなります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

なお，このオプションはデフォルトではチェックされていません。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

(3) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

5.5 リンク・マップ

この節では，ld850 の出力するリンク・マップについて説明します。

リンク・マップとは，リンク結果の情報が書かれたもので，セクションの配置アドレスなどの情報を知ることができます。リンク・マップの出力方法は，次のようになります。

5.5.1 コマンド・ライン上から ld850 起動する場合

-m オプションを指定すると，リンク終了時に標準出力にリンク・マップを表示します。また，-mo オプションを指定すると，CA850 Ver.2.60 以前の旧形式での表示となります。ファイルに出力したい場合は，-m=*file* オプション，または -mo=*file* オプションでファイル名を指定して出力します。

5.5.2 PM+ から起動する場合

[ツール] メニュー→ [リンカオプションの設定] で [ファイル] タブにおいて “ リンクマップファイル [-m=] ” にファイル出力指定されている場合，または [オプション] タブにおいて “ リンクマップの表示 [-m] ” がチェックされている場合に出力されます。出力先は，ファイル出力指定されているときは，指定されたファイルに，ファイル出力指定されていないときは，PM+ のアウトプット・ウインドウになります。

5.5.3 リンク・マップの出力例

次にリンク・マップの出力例を示します。例にあるオブジェクトをリンクした場合に出力されるリンクマップの一例を，図 5 － 12 に示します。

【オブジェクト】

crtN.o

main.o

func.o

libc.a（標準ライブラリ）

【リンク・マップの出力】

図 5 － 12 リンク・マップの出力例

```
                    ********** MEMORY ALLOCATION MAP **********

OUTPUT          SEGMENT         VIRTUAL         SIZE(16)        SIZE(10)
SEGMENT         ATTRIBUTE       ADDRESS

TEXT            RX              0x00000000      0x00000082            130
DATA            RW              0x00000088      0x00000018             24
 (1)            (2)               (3)             (4)             (5)

                       ******** LINK EDITOR ALLOCATION MAP ********

 OUTPUT          INPUT           VIRTUAL         SIZE            INPUT
 SECTION         SECTION         ADDRESS                         FILE

 .text                           0x00000000      0x00000082
                 .text           0x00000000      0x0000001a      crtN.o
                 .text           0x0000001c      0x0000002c      main.o
                 .text           0x00000048      0x00000018      func.o
                 .text           0x00000060      0x00000022      strcmp.o(../lib850/

 .sdata                          0x00000088      0x0000000e
                 .sdata          0x00000088      0x0000000e      main.o

 .sbss                           0x00000098      0x00000008
                 .sbss           0x00000098      0x00000004      func.o
                 .sbss           0x0000009c      0x00000004      *(GpCommon)*

  (6)            (7)               (8)             (9)             (10)
```

リンク・マップの “ MEMORY ALLOCATION MAP ” には，次の情報が表示されます。

(1) 出力セグメント

生成されるオブジェクト・ファイルを構成する出力セグメントの名前（出力セグメントの名前は格納されません）

(2) セグメント属性

R	読み出し可能
W	書き込み可能
X	実行可能

(3) アドレス

出力セグメントの先頭アドレス

(4) サイズ（16 進数）

セクション間の整列条件やアラインホールを含めたメモリのサイズ（16 進数）

(5) サイズ（10 進数）

セクション間の整列条件やアラインホールを含めたメモリのサイズ（10 進数）

リンク・マップの “ LINK EDITOR ALLOCATION MAP ” には，次の情報が表示されます。

(6) 出力セクション

ロード・モジュールに出力されたセクション名（12 文字まで表示）

(7) 入力セクション

出力セクションを構成する入力セクション名（12 文字まで表示）

(8) アドレス

出力セクション，または入力セクションの先頭アドレス

(9) サイズ

出力セクション，または入力セクションのサイズ

(10) 入力ファイル

入力セクションの属しているオブジェクト・ファイル名

アセンブリ言語で .comm 疑似命令を用いて領域確保した場合，全ファイル共通の領域となり，そのセクションは“ *(Common)* ”，または“ *(GpCommon)* ”と表示されます。入力セクションの属しているオブジェクト・ファイルがアーカイブ・ファイル（ライブラリ）内のオブジェクト・ファイルである場合，

- オブジェクト・ファイル名（アーカイブ・ファイル名）

の形式で，アーカイブ・ファイルの名前がフルパスで表示されます。

なお，-mo オプションを指定して，CA850 Ver.2.60 以前の旧形式での表示を指定した場合，ld850 によって作られたセクション，および as850 によって作られる .symtab，.strtab，および .shstrtab などのセクションに対しては *(nil)* が表示されます。

備考 *(nil)* 表示について

.sbss，sdata などのデータ領域に *(nil)* が表示されているものがあります。これは「グローバルに宣言された，初期値を持たない変数が配置されている」ことを指しています。これは，別のファイルに同名の変数があっても，最終的にロード・モジュールに集約されるため，変数の存在するファイル名が不明となってしまうため，リンク・マップ上では *(nil)* という表示になっています。

ただし，#pragma section “ data ”命令などを使用して，初期値なしデータを宣言した場合は，存在箇所が明らかになるため，*(nil)* ではなくファイル名が表示されます。

5.6 フラッシュ／外付け ROM 再リンク機能

5.6.1 再リンク機能とは

システムによっては，フラッシュ ROM や，着脱可能な ROM を搭載していることがあります。

フラッシュ ROM の場合は，書かれてある内容を書き換えたり，着脱可能な ROM の場合は，新しく書き換えた ROM 自体を取り替えることによって，プログラムのバージョン・アップ等を行います。

プログラムの一部でも変更する場合，基本的にプロジェクトそのものを再構築，つまり，“ リビルド ” して作成し直すことになります。しかし，バージョン・アップしたい箇所が，フラッシュ ROM や外付け ROM だけに限られている場合は，再構築しないで済むと便利です。また，ブート部分は内蔵 ROM などに固定され，書き換え対象のフラッシュ ROM との間に関数呼び出しがある場合，フラッシュ ROM 内の関数を修正することにより，関数の先頭アドレスがずれてしまうと，関数呼び出しが正常に行えなくなってしまいます。

このような状況を防ぎ，正常に関数呼び出しの実現をするのが“ フラッシュ／外付け ROM 再リンク機能 ”（以下 “ 再リンク機能 ”）です。

実現方法の概略は，次のようになります。

(1) フラッシュ ROM 領域に，フラッシュ ROM 領域内の関数群への分岐命令が書かれてある “ 分岐テーブル ” を用意する

(2) ブート領域から，フラッシュ ROM 領域内の関数をコールするとき，いったんフラッシュ ROM 領域の分岐テーブルへジャンプし，その後，目的の関数への分岐命令を実行してジャンプする

これらの仕組みをユーザで用意して実現することもできますが，この “ 再リンク機能 ” を用いると比較的簡単に実現できます。

ただし，この機能を使う上で，固定 ROM 側を作成した時点で，フラッシュ ROM 側の呼び出す関数は決定している必要があります。あくまでも，フラッシュ ROM 側の関数に変更があっても，固定 ROM 側からその関数を問題なく呼び出すことができるようにする仕組みです。

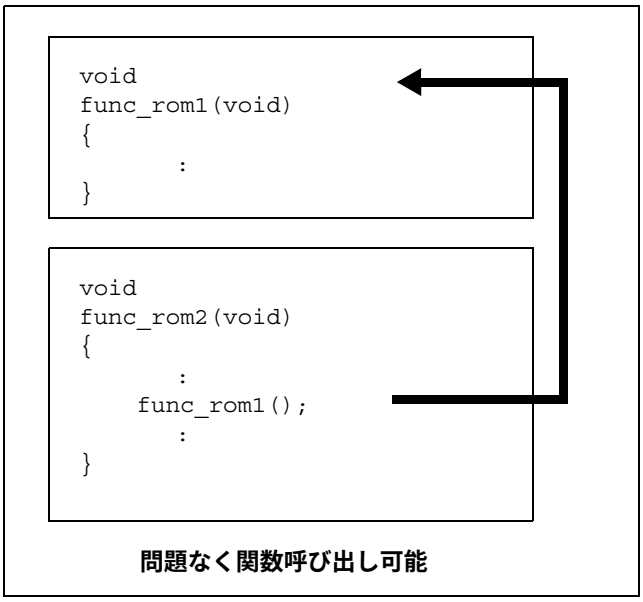
5.6.2 再リンク機能のイメージ

再リンク機能を利用したときの，関数呼び出しのイメージは次のようになります。

(1) 固定 ROM 内から固定 ROM 内の関数を呼び出すとき

固定 ROM に焼き付ける前に，すでにアドレス解決ができていることなので，問題なく関数呼び出しができます。

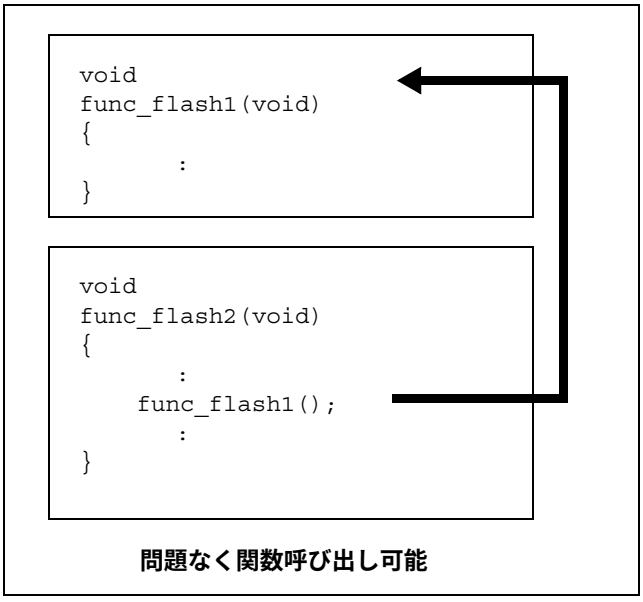
図 5 － 13 固定 ROM 内



(2) フラッシュ ROM 内からフラッシュ ROM 内の関数を呼び出すとき

フラッシュ ROM 内ではアドレス解決ができていることなので，問題なく関数呼び出しができます。

図 5 － 14 フラッシュ ROM 内

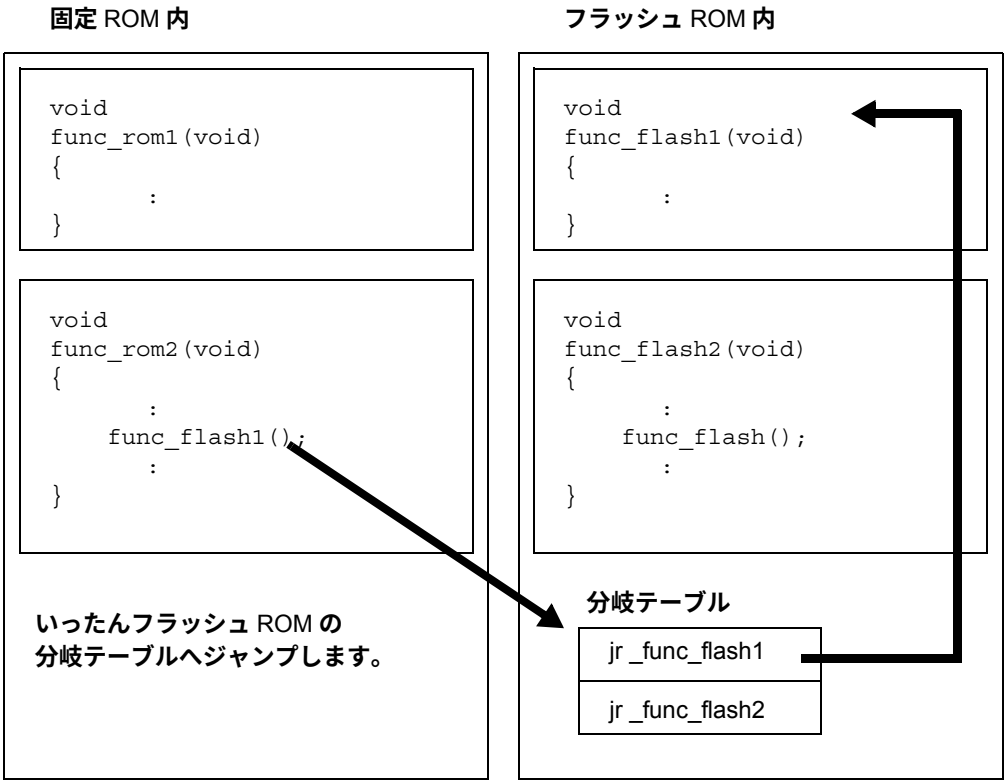


(3) 固定 ROM 内からフラッシュ ROM 内の関数を呼び出すとき

固定 ROM 内からフラッシュ ROM 内にある関数を呼び出すとき，固定 ROM 内からは，フラッシュ ROM 内の関数サイズ等の変更により，アドレスがわかりません。つまり，フラッシュ ROM 内の関数を直接呼び出すことができません。これを解決するため，いったんフラッシュ ROM 内の分岐テーブルへジャンプします。

そのテーブルから該当する関数へのジャンプ命令を実行し，目的の関数へジャンプします。

図 5 － 15 固定 ROM 内からフラッシュ ROM 内



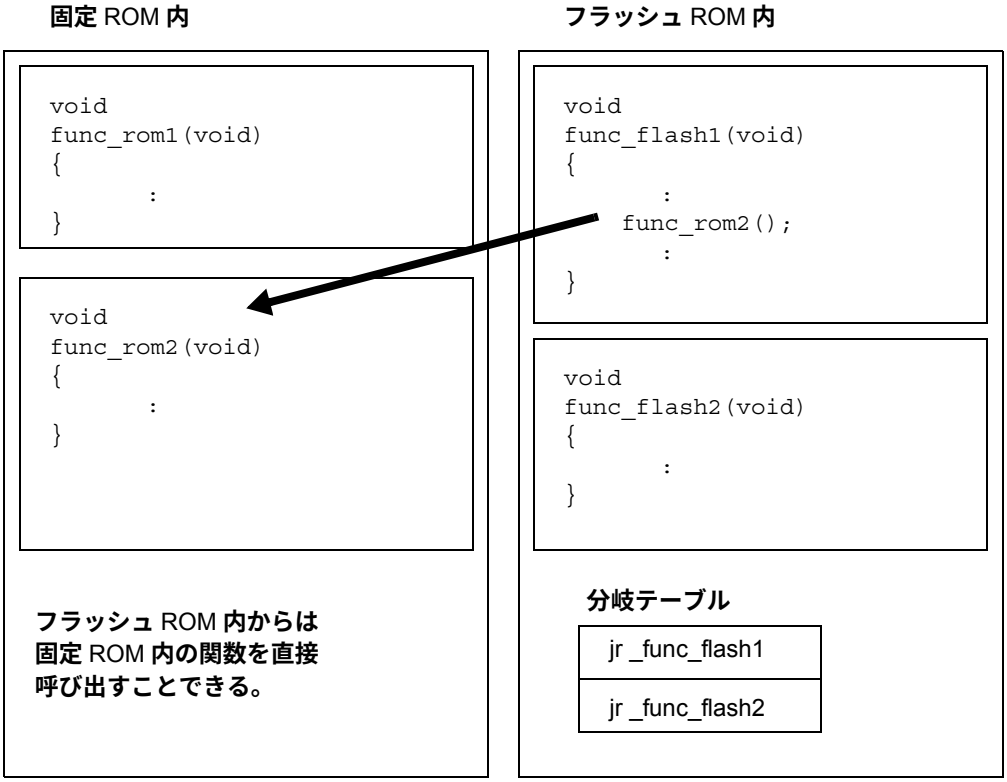
また，関数と同様に，外部変数の参照の可否にも関係します。

フラッシュ ROM 内に定義されているグローバル変数は，固定 ROM 内から参照することはできません。そのため，固定 ROM 内，フラッシュ ROM 内それぞれで同じ名前の外部変数を定義することができます。その外部変数に対する参照は，それぞれの領域内からの参照のみになります。

(4) フラッシュ ROM 内から固定 ROM 内の関数を呼び出すとき

フラッシュ ROM 内から固定 ROM 内にある関数を呼び出すとき，固定 ROM 内の内容は変わらないので，フラッシュ ROM 内からは固定 ROM 内にある関数を直接呼び出すことができます。

図 5 － 16 フラッシュ ROM 内から固定 ROM 内



また，関数と同様に，外部変数の参照の可否にも関係します。固定 ROM 内に定義されているグローバル変数は，フラッシュ ROM 内から参照することができます。

5.6.3 再リンク機能の実現方法

再リンク機能を実現する具体的な方法について説明します。

(1) PM+ のプロジェクト

再リンク機能を実現する場合，“ 固定 ROM 側 ” と “ フラッシュ ROM 側 ” を別々に作成することになります。つまり，一度固定 ROM 側を作成したあと（ROM に焼かれたのち）は，フラッシュ ROM 側だけを変更することになります。そのため，PM+ でプロジェクトを作成するときは，次のように分けて作成してください。

(a) 固定 ROM 側に配置するプロジェクト

(b) フラッシュ ROM 側に配置するプロジェクト（今後変更することがあるプロジェクト）

また，“ スタート・アップ・ルーチン ”，および “ リンク・ディレクティブ・ファイル ” も，それぞれのプロジェクト用に別々に用意します。

(2) .ext_func 疑似命令

固定 ROM 側から，フラッシュ ROM 側の関数を呼び出したい場合，まず，固定 ROM 側に .ext_func 疑似命令を使って“ 呼び出す関数名（ラベル名）と ID 番号 ”を付けます。.ext_func疑似命令の書式は次のようになります。

```
.ext_func 関数名 , ID 番号
```

ID 番号は正数で指定します。また，“ 同じ関数名で異なる ID 番号を指定 ” したり “ 異なる関数名に同じ ID 番号を指定 ” したりすることはできません。

固定 ROM 領域側に，.ext_func 疑似命令を使ってフラッシュ ROM 領域側にある関数名を指定すると，分岐テーブル（ext_table）が作成されます。この ext_table のアドレスはユーザで指定します。

指定方法は “ 内部 ROM 側のロード・モジュール ” を作成するとき，および “ フラッシュ ROM 側のロード・モジュール ” を作成するとき，それぞれのリンカ・オプション “ -ext_table ” で次のように指定します。

```
-ext_table 指定するアドレス
```

関数本体へ分岐するときは，作成した分岐テーブルの先頭から，ID 番号によるオフセット参照をすることによって実際の関数アドレスを取得し，そして分岐することになります。

例

```
func_flash0()
func_flash1()
func_flash2()
```

上記 3 つの C 関数がフラッシュ ROM に配置されていて，これらを固定 ROM 側から呼び出したい場合，固定 ROM 側にアセンブラで次のように記述します。

```
       .ext_func _func_flash0, 0
       .ext_func _func_flash1, 1
       .ext_func _func_flash2, 2
```

C 言語ソース・ファイル内に書くときは，#pragma asm ～ #pragma endasm **指令，または，**__asm() **命令を使って記述します。**#pragma asm ～ #pragma endasm **指令を使った例は，次のようになります。**

```
#pragma asm
       .ext_func _func_flash0, 0
       .ext_func _func_flash1, 1
       .ext_func _func_flash2, 2
#pragma endasm
```

なお，これらの .ext_func **疑似命令群の記述は，記述漏れやソース間の矛盾が生じることを防ぐため，つまり，“ 同じ関数名で異なる** ID **番号を指定 ” したり “ 異なる関数名に同じ** ID **番号を指定 ” というような間違いを防ぐため，**1 **つのファイルにまとめて，すべてのソースに** .include **疑似命令で（**C **言語で記述するときは，**#include **命令で）インクルードすることを推奨します。**

上記のように #pragma asm ～ #pragma endasm **指令を使ったファイルをインクルードすると，コンパイル時に次のメッセージが出ますが，無視してください（または，“ 個別の警告 ” で出力しないように設定してください）。**

```
W2244: '#pragma asm' used out of function is not supported completely.
```

再リンク機能のイメージは，次のようになります。

<table>
<tr><th>ユーザが記述するアセンブリ言語ソース</th><th>リンク後のアセンブラ・イメージ</th></tr>
<tr><td>

```
[ext_table.inc]
       .ext_func _func_flash0, 0
       .ext_func _func_flash1, 1
       .ext_func _func_flash2, 2
```

</td><td></td></tr>
<tr><td>

```
[rom.s]
       .include "ext_table.inc"
       .extern _func_flash0
       .extern _func_flash1
       .extern _func_flash2
       jarl   _func_flash0, lp
       jarl   _func_flash1, lp
       jarl   _func_flash2, lp
```

</td><td>

```
[rom.out]
       .extern __ext_table_head
       jarl   __ext_table_head+0x4*0,lp
       jarl   __ext_table_head+0x4*1,lp
       jarl   __ext_table_head+0x4*2,lp
```

</td></tr>
<tr><td rowspan="2">

```
[flash.s]
       include "ext_table.inc"
       .globl _func_flash0
       .globl _func_flash2
_func_flash0:
             :
       jmp    [lp]

       .globl _func_flash1
_func_flash1:
             :
       jmp    [lp]

_func_flash2:
             :
       jmp    [lp]
```

</td><td>

```
[flash.o]
#（分岐テーブル）
       .section “.ext_table”, text
       .globl __ext_table_head
       .extern _func_flash0
       .extern _func_flash1
       .extern _func_flash2
__ext_table_head:
       jr     _func_flash0
       jr     _func_flash1
       jr     _func_flash2
```

</td></tr>
<tr><td>

```
#（関数本体）
       .globl _func_flash0
       _func_flash0:
       :
       jmp    [lp]

       .globl _func_flash1
_func_flash1:
             :
       jmp    [lp]

       .globl _func_flash2
_func_flash2:
             :
       jmp    [lp]
```

</td></tr>
</table>

このように .ext_func 疑似命令を指定すると，ext_table というシンボルでテーブルが生成され，その先頭シンボルが “ __ext_table_head ” になります。

固定 ROM 側の“ jarl _func_flash0, lp ”というコードは，__ext_table_head からのオフセットで _func_flash0 のアドレスを取得し，jarl 命令で関数本体へジャンプします。

(3) スタート・アップ・ルーチン

固定 ROM 側のスタート・アップ・ルーチンと，フラッシュ ROM 側のスタート・アップ・ルーチンは，それぞれに用意します。それぞれのスタート・アップ・ルーチンでしなくてはならない処理は，次のようになります。

(a) 固定 ROM 側で tp，gp，ep 値をセットする

(b) 固定 ROM 側で使用する RAM 領域を初期化するため，_rcopy 関数を呼び出す

(c) 固定 ROM 側からフラッシュ ROM 側のスタート・アップ・ルーチンへ分岐する

(d) フラッシュ ROM 側で使用する RAM 領域を初期化するため，_rcopy 関数を呼び出す

(e) フラッシュ ROM 側の処理へ移行

tp，gp，ep を固定 ROM 内で使用しない場合は，値のセットをフラッシュ ROM で行っても問題ありません。また，_rcopy 関数を使用して初期値データのコピーを行う際は，ロード・モジュールに対し，ROM 化プロセッサによる“ ROM 化 ”を行っている必要があります。rompsec セクションの先頭シンボルを持った rompcrt.o を用意し，リンカ・オプション“ -lr ”を指定してリンクします。これによって作成されたパッキング・セクションを用いて，_rcopy 関数で初期値ありデータをコピーします（「第 6 章 ROM 化プロセッサ」参照）。

tp，gp，ep 値ですが，これらは固定 ROM 側，およびフラッシュ ROM 側で同じアドレス値を使用することを推奨します。異なる値にすることも可能ですが，その場合は固定 ROM 側とフラッシュ ROM 側の命令コードを行き来するたびに，値を設定しなおす必要が出てきます。

固定 ROM 側	フラッシュ ROM 側

`__start:`
 	
`mov #__tp_TEXT, tp`
`mov #__gp_DATA, gp`
`mov #__ep_DATA, ep`
 	
`:`
 	
`# 固定 ROM 側の main 関数へ`
`# main 関数という名前にこだわる必要はない`
`jarl _main, lp`
 	
`.ext_func _flash_start 3`
`jr __flash_start`
	`.ext_func _flash_start 3`
	`jr __flash_start`
	
	`__flash_start:`
	
	`:`
	
	`# フラッシュ ROM 側の main 関数へ`
	`jarl _main, lp`

`extern unsigned long _S_romp;`
 	
`void main(void)`
`{`
`_rcopy(&_S_romp, -1);`
`:`
`}`
	`extern unsigned long _S_romp;`
	
	`void main(void)`
	`{`
	`_rcopy(&_S_romp, -1);`
	`:`
	`}`

(4) リンク・ディレクティブ・ファイルの記述

固定 ROM 側のプロジェクトと，フラッシュ ROM 側のプロジェクトで，それぞれリンク・ディレクティブを持ちます。リンク・ディレクティブ・ファイルを記述する際の注意事項は次のとおりです。

(a) RAM 領域に置くセクションのアドレスは,固定 ROM 側とフラッシュROM 側でオーバラップしていても，プロジェクトが異なるので，リンカ等でエラーを出力することができません。つまり，オーバラップさせることが可能です。固定 ROM 側とフラッシュ ROM 側で同時に参照する必要のある RAM 領域は，オーバラップさせないようにアドレス指定する必要があります。

(b) tp，gp，ep 値ですが，これらは固定 ROM 側，およびフラッシュ ROM 側で同じアドレス値を使用することを推奨します。異なる値にすることも可能ですが，その場合は固定 ROM 側とフラッシュ ROM 側の命令コードを行き来するたびに，値を設定しなおす必要が出てきます。

(c) 分岐テーブル（ext_table）に関するリンク・ディレクティブの記述は必要ありません。リンカ・オプション “ -ext_table ” で指定されたアドレスに自動的に配置されます。
ただし，次の点に注意が必要です。

- -ext_table で指定されたアドレスに，分岐テーブルのサイズ分の空き領域があった場合，そのまま配置されます。他のセグメントへの影響はありません。このようにするのが最も理想的です。
- -ext_table で指定されたアドレスに，分岐テーブルのサイズ分の空き領域がなかった場合，エラーになります。たとえば -ext_table で指定したアドレスは “ アドレス指定された TEXT セグメント内 ” で，すでにコードが配置されている場合などです。これに該当する例は，次のようになります。

分岐テーブルのアドレス指定

```
-ext_table 0x500
```

リンク・ディレクティブ・ファイル（の一部）

```
TEXT : !LOAD ?RX V0x400 {
       .text = $PROGBITS ?AX .text;
};
```

（TEXT セグメントのサイズが 0x100 バイト以上あるとします）

このとき，分岐テーブルは 0x500 番地に割り付けることができないため，リンク時にエラーになります。-ext_table で指定する値を変更してください。

- 再リンク機能を使う前は，-ext_table で指定したアドレスには，他のセグメントが割り当てられていたが，リンク・ディレクティブには，そのセグメントのアドレス指定がなかった場合，-ext_table で指定されたアドレスには分岐テーブルが配置され，元あったセグメントは，分岐テーブルの後ろに移動します。
  ただし，セグメントがずれたことにより，さらに後ろの“アドレス指定されたセグメント”に重なった場合は，エラーになります。

分岐テーブルのアドレス指定

```
-ext_table 0x500
```

リンク・ディレクティブ・ファイル（の一部）

```
TEXT : !LOAD ?RX {
       .text = $PROGBITS ?AX .text;
};
```

（TEXT セグメントよりも前のセグメントから続きで，0x500 番地から TEXT セグメントが割り当てられていたとします）

このとき，TEXT セグメントにはアドレス指定がないため，分岐テーブルは 0x500 番地に割り付けられ，TEXT セグメントは分岐テーブルの後ろに割り付けられます。

(5) アセンブラ・オプションとリンカ・オプション

フラッシュ ROM 側のオブジェクトをアセンブルする際は，アセンブラ・オプション“ -zf ”を指定します。

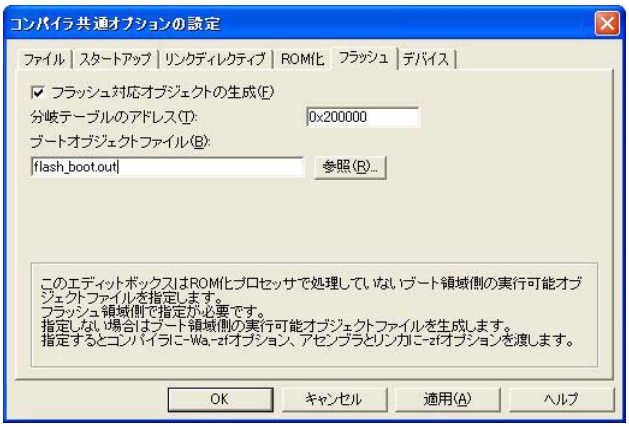
なお，as850 単体で起動するときは，このオプションを指定しますが，PM+ から起動するときは，コンパイラ共通オプションで“ フラッシュ対応オブジェクトの生成 ”にチェックし“ ブートオブジェクトファイル ”で，固定 ROM 側のオブジェクト・ファイルを指定すると，自動的にアセンブル・オプションとして“ -zf ”が付加されます。

ここで指定するものは，ld850 が出力したオブジェクトです。romp850 が出力したオブジェクトを指定すると，エラーとなるので注意してください。

また，リンカ・オプションとして，“ 分岐テーブル（ext_table）のアドレスの指定（-ext_table オプション）”を指定してください。

ここで指定するアドレスは，フラッシュ ROM 内のアドレスを指定してください。分岐テーブルを 0x200000 番地とした場合の例は，次のようになります。

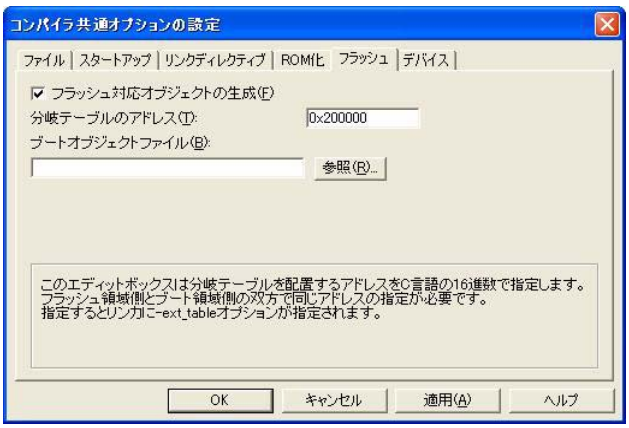
図 5 － 17 フラッシュ ROM 側のコンパイラ共通オプション設定



固定 ROM 側のプロジェクトに対しても“ フラッシュ対応オブジェクトの生成 ”にチェックしますが，“ ブートオブジェクトファイル（-zf オプション）”は指定しないでください。

なお，“ 分岐テーブル（ext_table）のアドレスの指定（-ext_table オプション）”に関しては，フラッシュ ROM 側と同じアドレスを指定してください。固定 ROM 側のオプションの例は，次のようになります。

図 5 － 18 固定 ROM 側のコンパイラ共通オプションの設定



(6) .ext_ent_size 疑似命令

フラッシュ ROM 内にある分岐テーブルから，実際の関数を呼び出すとき，デフォルトでは，次のように jr 命令による分岐命令が出力されます。

```
__ext_table_head:
       jr     _func_flash0
       jr     _func_flash1
       jr     _func_flash2
```

しかし，アーキテクチャ上，jr 命令では 22 ビット範囲内（± 1M バイト範囲内）にしか分岐することができません。それ以上のアドレス，つまり，32 ビット空間すべてに分岐できるようにしたい場合，.ext_ent_size 疑似命令を追加で指定します。.ext_ent_size 疑似命令は書式は，次のようになります。

```
.ext_ent_size テーブルのエントリ・サイズ
```

エントリ・サイズで指定できる値は“ 4 ”“ 8 ”“ 10 ”のいずれかになります。ここでの“ テーブルのエントリ・サイズ ”とは“ 1 つの分岐処理に必要な命令サイズ ”ということになります。

デフォルトは“ 4 ”で，この場合は，次のように 4 バイト命令が配置されます。

```
jr     _flash_func0                        -- 4 バイト命令
```

“ 8 ”を指定すると，次のように合計で 8 バイトの命令が配置されます。

```
mov    #_flash_func0, r1                   -- 6 バイト命令
jmp    [r1]                                -- 2 バイト命令
```

“ 10 ”を指定すると，次のように合計で 10 バイトの命令が配置されます。

```
movhi  hi1(#_flash_func0), r0, r1          -- 4 バイト命令
movea  lo(#_flash_func0), r1, r1           -- 4 バイト命令
jmp    [r1]                                -- 2 バイト命令
```

8 バイト命令が使用できる（この命令セットに対応している）のは“ V850Ex ／ V850E2 コアの場合のみ ”です。

V850 で使用する場合は“ 10 ”を設定してください。また，V850 ／ V850Ex ／ V850E2 コア共通のオブジェクトを作成する場合（-cn オプション指定する場合）は，“ 10 ”で統一してください。

(7) ライブラリについて

固定 ROM やフラッシュ ROM からライブラリ関数を呼び出していた場合，ライブラリは呼び出した側のオブジェクトにリンクされます。たとえば，フラッシュ ROM 側にライブラリがリンクされていても，固定 ROM からも同じライブラリ関数を呼び出していた場合は，固定 ROM にも同じライブラリがリンクされます。つまり，ライブラリ関数を呼び出す場合は，固定 ROM とフラッシュ ROM との間で分岐は起こらないため，ライブラリ関数に対して .ext_func 疑似命令で関数指定する必要はありません。

ただし，“ 固定 ROM 側にリンクされたライブラリが，フラッシュ ROM 側の関数へ分岐する ” というような特殊な場合に関しては，.ext_func 疑似命令で関数指定する必要があります。

CA850 にパッケージされている “ 標準ライブラリ ” や “ 数学ライブラリ ” に関しては，.ext_func 疑似命令で関数指定する必要はありません。

(8) 割り込みハンドラについて

割り込みハンドラの呼び出し部分は，割り込みハンドラ・アドレスのある領域側に記述してください。
次の場合，割り込みハンドラ関数名に対しても，.ext_func 疑似命令で関数指定する必要があります。

- 割り込みハンドラ・アドレスは “ 固定 ROM 側 ”
- 割り込みハンドラ本体は “ フラッシュ ROM 側 ”

<table>
<tr><th>ユーザが記述するアセンブリ言語ソース</th><th>リンク後のアセンブラ・イメージ</th></tr>
<tr><td>

```
[ext_table.inc]
       .ext_func _int_flash0, 0
```

</td><td></td></tr>
<tr><td>

```
[rom.s]
       .include  "ext_table.inc"
       .extern _int_flash0
       .section "INT00", text
       jr     _int_flash0
```

</td><td>

```
[rom.out]
       .section "INT00", text
       jr     __ext_table_head+0x4*0,lp
```

</td></tr>
<tr><td rowspan="2">

```
[flash.s]
       .include "ext_table.inc"
       .globl _int_flash0
_int_flash0:
              :
       reti
```

</td><td>

```
[flash.o]
#（分岐テーブル）
       .section ".ext_table", text
       .globl __ext_table_head
       .extern _int_flash0
__ext_table_head:
       jr     _int_flash0
```

</td></tr>
<tr><td>

```
#（ハンドラ本体）
       .globl _int_flash0
_int_flash0:
              :
       reti
```

</td></tr>
</table>

5.7 補足事項

この章では，ld850 に関するその他の補足的な事項について説明します。

5.7.1 -A オプションの使い方

この項では，-A オプションの使い方について説明します。-A オプションは PM+ の場合，[リンカオプションの設定] ダイアログ上の [オプション] タブにある “ GP 情報の表示 [-A] ” を指定します。

-A

【機能】

ソース・ファイルのコンパイル時，およびアセンブル時に，ca850，および as850 に対して指定できる -G*num* オプションに対し，*num* に設定する値の目安となる情報を，

- コマンド・ラインで -A オプションを指定して起動した場合は標準出力
- PM+ で “ GP 情報の表示 [-A] ” を指定した場合は，アウトプット・ウインドウ

に表示します。

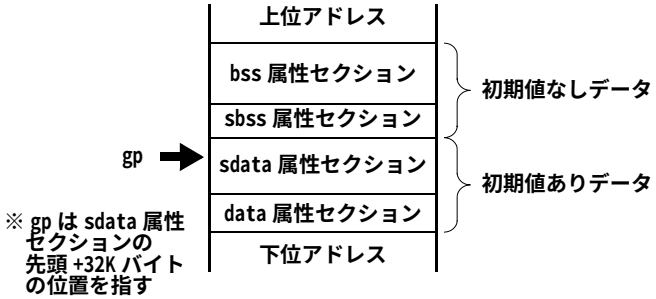
-G*num* オプションは「*num* バイト以下のデータを .sdata セクション，または .sbss セクションに配置する」オプションです。

ca850 および as850 は，sdata，sbss，data，bss 領域に配置するデータに対し，次のような規則に従ってコードを出力します。

まず gp レジスタから 1 命令でアクセスできる領域である “ sdata 属性セクション ”, “ sbss 属性セクション ” へ配置しようとします（初期値ありデータが sdata 属性セクションへ，初期値なしデータが sbss 属性セクションへ配置されます）。

これらの領域は gp と 16 ビットのディスプレイスメントを用いてアクセスするコードになるため，配置できる範囲は gp から± 32K バイト内に限られます。これらの領域に収まりきらなかった場合，gp レジスタから 2 命令でアクセスできる領域である “ data 属性セクション ”, “ bss 属性セクション ” へ配置しようとします（初期値ありデータが data 属性セクションへ，初期値なしデータが bss 属性セクションへ配置されます）。これらの領域は，ますアクセス領域のアドレス生成を行い，gp と 32 ビットのディスプレイスメントを用いてアクセスするコードを生成します。そのため，4G バイトすべての空間へアクセスが可能になります。

図 5 － 19 gp オフセット参照セクションのメモリ配置イメージ



したがって，1 命令でアクセス可能な sdata 属性セクション，sbss 属性セクションに，より多くのデータを割り付けた方が，実行効率やオブジェクト効率が良くなります。

データの割り付けは，C 言語ソースの場合は #pragma section 指令，アセンブラ言語ソースの場合は .section 疑似命令によって，ユーザが意図的に配置場所を指定する方法があります。

この方法の他に，sdata 属性セクション，sbss 属性セクションへ割り付けるデータのサイズの閾値を設け，そのサイズ以下のデータを sdata 属性セクション，sbss 属性セクションに配置する指定ができれば，ソース・プログラムに手を加えることなく，より多くのデータを配置することが可能になります。この指定をするのが ca850，as850 で指定する -G*num* オプションです。ここで *num* に指定する値は，データ・サイズになりますが，目安となる値がわかると便利です。

この情報を出力するのが，-A オプションです。

-A オプションが ld850 に指定された場合，-G*num* オプションの *num* の設定において目安として用いることのできる情報を出力します。

【出力情報の説明】

（-r オプションを指定しない）実行可能なオブジェクト・ファイルの生成時にこのオプションを指定した場合の出力情報の例と，（-r オプションを指定した）リロケータブルなオブジェクト・ファイルの生成時にこのオプションを指定した場合の出力情報の例を，次に示します。

【実行可能なオブジェクト・ファイルに対する出力情報の例】

```
        ******** LINK EDITOR GP INFORMATION ********
GP SYMBOL     SECTION       SECTION       SECTION       GP
NAME          NAME          SIZE(REAL)    SIZE(ASSUMED) NUMBER

_gp_DATA
              .sdata        0x000af10
                                          0x00002000    4       *OK*
                                          0x00003450    8       *OK*
                                          0x00004430    12      *OK*
                                          0x000050a8    16      *OK*
                                          0x00007b40    20      *OK*
                                          0x0000a010    24
                                          0x0000af10    32
              .sbss         0x00012050
                                          0x00000050    4       *OK*
                                          0x00002050    16      *OK*
                                          0x00007050    512     *OK*
                                          0x00010050    1024
 (a)          (b)           (c)           (d)           (e)     (f)
```

【リロケータブルなオブジェクト・ファイルに対する出力情報の例】

```
        ******** LINK EDITOR GP INFORMATION ********
GP SYMBOL     SECTION       SECTION       SECTION       GP
NAME          NAME          SIZE(REAL)    SIZE(ASSUMED) NUMBER

*(NOT AVAILABLE)*
              .sdata        0x000af10
                                          0x00002000    4       *OK*
                                          0x00003450    8       *OK*
                                          0x00004430    12      *OK*
                                          0x000050a8    16      *OK*
                                          0x00007b40    20      *OK*
                                          0x0000a010    24
                                          0x0000af10    32
              .sbss         0x00012050
                                          0x00000050    4       *OK*
                                          0x00002050    16      *OK*
              *GpCommon*    0x00010000
                                          0x00005000    512     *OK*
                                          0x00010000    1024
 (a)          (b)           (c)           (d)           (e)     (f)
```

出力情報の意味は，次のとおりです。

(a) グローバル・ポインタ・シンボルの名前

リンク時に用いられたグローバル・ポインタ・シンボルの名前。リロケータブルなオブジェクト・ファイルの場合，“ *(NOT AVAILABLE)* ” が表示されます。

(b) セクション名

データが割り付けられた sdata ／ sbss 属性セクションの名前。リロケータブルなオブジェクト・ファイルでは未定義外部シンボルのセクションへの割り付けを確定することはできないため，ld850 は，仮想的なセクション “ *GpCommon* ” を内部的に生成し，このセクションに一時的に割り付けます。

(c) セクションの実サイズ

データの整列によって生じるホールの領域などが考慮されている，“ セクションの実サイズ ”。

(d) 想定されるセクションのサイズ

この欄の右の欄で示される値を *num* として -G*num* オプションを指定して ca850 を起動した場合に想定されるセクションのサイズ。このサイズの計算では，各データに対し実際の整列条件は考慮せず一律に 4 バイト以上の整列条件を想定しているため，必ずしも実際に生成されるセクションの実サイズに一致するとはかぎりません。

(e) 想定された -G*num* オプションの *num* の値

この欄の左の欄で示される，“ 想定されるセクションのサイズ ” の計算において想定された，ca850，および as850 起動時における -G*num* オプションの *num* の値。

(f) 判定結果

この欄の左の欄で示される値を *num* として -G*num* オプションを指定し，ca850 を起動した場合にセクションのサイズが 15 ビット（0x0 ～ 0x7fff）の範囲に入るかどうかの判断結果[注]。入る場合 “ *OK* ” が表示され，入らない場合は何も表示されません。

注 CA850 では，通常，データの割り付けられるセクションは data ／ sdata ／ sbss ／ bss 属性セクションの順に下位のアドレスから割り付けられ，グローバル・ポインタ（gp）は sdata 属性セクションの先頭アドレス＋ 32K バイトを指すようスタート・アップ・モジュールなどにおいて設定されることが想定されているため，この判定で OK が出た場合 sdata ／ sbss 属性セクションは 16 ビットのディスプレースメントを用いて参照することのできるメモリの範囲に割り付けられていると考えることができます。

[注意事項]

- このオプションで出力される情報は，あくまで目安であり，たとえば，次のような場合，判定結果が正しくなくなる可能性があります。

(a) リンク・ディレクティブなどにおいて，ホールを生成するようなセクションの配置を指定した

(b) グローバル・ポインタ・シンボルに対し直接アドレスを指定した

(c) #pragma section 指令で，.sdata ／ .sbss セクションにデータを割り当てた

【使用例】

```
ld850 -A file1.o file2.o
```

file1.o と file2.o をリンクし，コンパイル時，およびアセンブル時に ca850，および as850 に対して指定することのできる -G*num* オプションの *num* の設定において目安として用いることのできる情報を標準出力に出力します。

5.7.2 アーカイブ・ファイル

アーカイブ・ファイルは，アーカイバ（ar850）において複数のオブジェクト・ファイルを結合することによって生成されます。

ld850 は，アーカイブ・ファイルが指定された時点において未解決な外部参照についてアーカイブ・ファイルを検索し[注 1]，必要とされるオブジェクト・ファイルのみをリンクします。

アーカイブ・ファイルは，リンク・ディレクティブのマッピング・ディレクティブでも指定できます。マッピング・ディレクティブにおいて指定された場合も，その指定された時点において未解決な外部参照について検索され，必要とされるオブジェクト・ファイル[注 2] のみがリンクされます。

注 1 アーカイブ・ファイルは，それが含んでいる各オブジェクト・ファイルに属すシンボルのシンボル・テーブルを持っており，そのアーカイブ・ファイルにより未解決な外部参照が解決されなくなるまで繰り返し検索されます。

注 2 参照されているシンボルが定義されているオブジェクト・ファイルです。

5.7.3 予約シンボル

ld850 は，リンクの処理において，各出力セクションの先頭アドレス，各出力セクションの終端を越える最初のアドレス，および生成された実行可能なオブジェクト・ファイルの終端を越える最初のアドレス値を値として持つ予約シンボルを生成します。

ユーザがこれらの予約シンボルと同名のシンボルを定義した場合，ld850 は定義されたシンボルを用い，独自に生成することはしません。

セクションの先頭アドレス値を値として持つ予約シンボルとしては，その出力セクションの名前の頭に“__s”を付けることによって構成される名前のシンボルが用いられます。

ただし，そのセクション名が“.”で始まっている場合，その“.”を取った後ろの名前の頭に“__s”を付けることによって構成される名前のシンボルが用いられます。セクションの終端を越える最初のアドレス値を値として持つ予約シンボルとしては，その出力セクションの名前の頭に“__e”を付けることによって構成される名前のシンボルが用いられます。

ただし，そのセクション名が“.”で始まっている場合，その“.”を取った後ろの名前の頭に“__e”を付けることによって構成される名前のシンボルが用いられます。生成された実行可能なオブジェクト・ファイルの終端を越える最初のアドレス値を値として持つ予約シンボルとしては，__end が用いられます。

ld850 の用いるデフォルトのリンク・ディレクティブでは出力セクションとして次の予約セクションが用いられています。

表 5 － 2 予約セクション

.text，	.pro_epi_runtime，	.data，	.sdata，
.sbss，	.bss，	.sconst，	.const，
.sedata，	.sebss，	.sidata，	.sibss，
.tidata，	.tibss，	.tidata.byte，	.tibss.byte，
.tidata.word，	.tibss.word		

このため，ld850 は通常次に示した予約シンボルを生成することになります。

表 5 － 3 通常のオブジェクト・ファイルにおける特殊シンボル

__end，	__ebss，	__econst，	__edata，
__epro_epi_runtime，	__esbss，	__esconst，	__esdata，
__esebss，	__esedata，	__esibss，	__esidata，
__etext，	__etibss，	__etibss.byte，	__etibss.word，
__etidata，	__etidata.byte，	__etidata.word，	__sbss，
__sconst，	__sdata，	__spro_epi_runtime，	__ssbss，
__ssconst，	__ssdata，	__ssebss，	__ssedata，
__ssibss，	__ssidata，	__stibss，	__stibss.byte，
__stibss.word，	__stidata，	__stidata.byte，	__stidata.word

注意 生成するシンボルは，表 5 － 3 のうち，リンク処理後の実行形式ファイルにセクションが存在するもののみとなります。ld850 では，リンク・ディレクティブ・ファイルに対してマッピング・ディレクティブを記述しても，実際に割り付けられるセクションが存在しなければ，セクションは存在しないものとして扱います。

5.7.4 期待したセクションに割り付けられない場合

ディレクティブ・ファイル内で，セクションに割り付けるオブジェクト・ファイルやアーカイブ・ファイルを指定しても，ファイル名の記述の仕方によってはそれらが期待したセクションに割り付けられないことがあります。その場合，リンク・マップ（-m）を参照しながら，リンク・マップで表示されているファイル名と，パス名も含めてまったく同じ名前でディレクティブ・ファイルに指定し，再リンクしてください。

5.7.5 V850 コアと V850Ex コア

V850Ex は，他の V850 コア・マイクロプロセッサに対して上位互換です。V850 コアで使用していたソース・プログラムを，V850Ex で使用できます。その際，V850 コア・オブジェクトは，as850 のオプションにより，コア内共通のオブジェクト・ファイルとして作成してください。

なお，“ V850Ex 内共通 ” として作成したオブジェクト・ファイルは，V850Ex 以外のオブジェクト・ファイルとリンクすることはできません。

詳細は，「4.7.1 マジック・ナンバ」を参照してください。

5.7.6 V850 コアと V850E2 コア

V850E2 は，他の V850 コア・マイクロプロセッサに対して上位互換です。V850 コアで使用していたソース・プログラムを，V850E2 で使用できます。その際，V850 コア・オブジェクトは，as850 のオプションにより，コア内共通のオブジェクト・ファイルとして作成してください。

なお，“ V850E2 内共通 ” として作成したオブジェクト・ファイルは，V850E2 以外のオブジェクト・ファイルとリンクすることはできません。

詳細は，「4.7.1 マジック・ナンバ」を参照してください。

5.7.7 数学ライブラリ

数学ライブラリ関数をプログラム中に使用し，リンク時に数学ライブラリ（libm.a）をリンクしても，undefined symbol などのエラーが出ることがあります。これは標準ライブラリなどとのリンク順番に関係していることがあります。ANSI 準拠の順番でリンクする必要がありますので，標準ライブラリを最後にリンクしてください。特にコマンド・ラインからリンカを起動するときには注意してください。具体的には -lm，-lc の順番にオプションを記述してください。

5.7.8 main 関数

main 関数を作らずにリンクをした場合，_main シンボルが undefined symbol エラーとして出力されることがあります。特にスタート・アップ・ルーチンを独自に指定せず，デフォルトのスタート・アップ・ルーチン（crtN.o，crtE.o【V850E】）がリンクされた場合，またはパッケージに用意されている crtN.s，crtE.s をそのままアセンブル，リンクして使用した場合に発生します。これは crtN.s や crtE.s の最後の方に書かれてある「jarl _main, lp」というコードが原因です。main 関数が必要ない場合，ここを独自に書き換え，再アセンブルして作成したオブジェクトをスタート・アップ・ルーチンとしてください。また，リアルタイム OS を使用したアプリケーションの場合，通常 main 関数はありません。リアルタイム OS のサンプルとして提供されているスタート・アップ・ルーチンを使用してください。

5.7.9 プロローグ／エピローグ・ランタイム・ライブラリ

プロローグ／エピローグ・ランタイム・ライブラリは，専用の .pro_epi_runtime セクションに配置する必要があります。配置していない場合，次のメッセージを出力し，リンクを中止します。

```
F4286 : section ".pro_epi_runtime" must be specified in link directive.
```

リンク・ディレクティブ・ファイルを指定している場合には，.text セクションの手前にマッピング・ディレクティブを記述してください。

```
.pro_epi_runtime = $PROGBITS ?AX .pro_epi_runtime;
.text = $PROGBITS ?AX;
```

.pro_epi_runtime セクションを .text セクションの後ろに配置すると，ROM 化の際，パッキングされたセクションのデフォルト動作での配置位置と重なります。.text セクションの手前に配置することを推奨します。リンク・ディレクティブ・ファイルを指定しない場合には，.text セクションの手前にリンクします。

[注意事項]

- プロローグ／エピローグ・ランタイム・ライブラリは，標準ライブラリ libc.a に含まれています。
- .pro_epi_runtime セクションは通常のセクションと異なり，入力セクション名が固定されており，専用のセクションだけが配置されます。
- .pro_epi_runtime セクションを .text セクションの後ろに配置すると，ROM 化する場合にパッキングされたセクションのデフォルト動作での配置位置と重なります。.text セクションの手前に配置してください。
- V850Ex ／ V850E2 コアのデバイス指定時にはプロローグ／エピローグ・ランタイム・ライブラリは，callt 命令を使用しています。スタート・アップ・ルーチンで CTBP の設定を行ってください。

5.7.10 ROM 化のためのリンク

ROM 化を行う場合は，パッキング・セクション領域を考慮したリンク・ディレクティブを記述する必要があります。詳細については，「第 6 章 ROM 化プロセッサ」を参照してください。

デフォルトのリンク・ディレクティブを使用し，CONST セグメントを利用する場合，ROM 化がうまくできません。それは，デフォルトのリンク・ディレクティブでは，CONST セグメントを TEXT セグメントの直後に配置しているため，ROM 化プロセッサがデフォルトの動作によって，パッキング・セクション（rompsec セクション）と CONST セグメントが重なってしまうためです。パッケージに添付されているサンプル・ディレクティブ[注]を参考にし，次のいずれかの対応を行ってください。

注 “インストール・フォルダ \smp850\ca850 ”に格納されている“ v850def.dir ／ v850def2.dir ／ v850def3.dir ” です。

v850def.dir	内蔵 ROM ／ RAM，外部 RAM を使ったサンプル
v850def2.dir	内蔵 ROM ／ RAM だけを使ったサンプル
v850def3.dir	内蔵 ROM ／ RAM，外部 RAM，内蔵命令 RAM を使ったサンプル （V850E/ME2 など）

また,使用するマイクロプロセッサにあわせたメモリ配置を行う必要があります。CONST セグメントを TEXT セグメントの手前に配置します。

```
CONST : !LOAD ?R{
        .const = $PROGBITS ?A .const;
};

TEXT : !LOAD ?RX{
        .text = $PROGBITS ?AX;
};
```

TEXT セグメントの後ろにパッキング・セクションの領域（「第 6 章 ROM 化プロセッサ」参照）を確保し，その後ろに CONST セグメントを配置します。

```
TEXT : !LOAD ?RX{
        .text = $PROGBITS ?AX;
};

        [ パッキング・セクションの領域 ]

CONST : !LOAD ?R V0x200000{        ←パッキング・セクションを考慮したアドレスを指定
        .const = $PROGBITS ?A .const;
};
```

5.7.11 プログラマブル周辺 I/O レジスタ

プログラマブル周辺 I/O レジスタ機能を使用するアプリケーション・プログラムの場合，アセンブル時に予約セクションの .bpc セクションが出力されます。ld850 は，リンクの入力オブジェクト・ファイルに .bpc セクションが存在する場合，BPC 値として指定されている値のチェックを行います。入力オブジェクト・ファイル間で値が統一されていない場合，リンカは次のようなエラー・メッセージを出力し，リンク処理を中断します。

```
F4457: input files have different BPC value.
0x00001234    file1.o
0x00001234    file2.o
0x00001235    file3.o
*(none)*      file4.o
```

上記の場合，file3.o で設定されている値が異なるため，エラーとなっています。
なお，プログラマブル周辺 I/O レジスタを参照していないオブジェクトは，チェックの対象とはなりません。
上記の file4.o のように “ *(none)* ” と表示されます。

BPC 値のチェックでエラーがなかった場合，セクション・タイプ SHT_PROGBITS，セクション属性 none，セクション・サイズ 0x4 の .bpc セクションが生成されます。.bpc セクションには，BPC 値を既定ビット数分シフトして得られる，プログラマブル I/O レジスタ領域の先頭アドレスが格納されます。

例

V850E/IA1 使用時に BPC 値を “ 0x1234 ” と指定した場合，プログラマブル周辺 I/O レジスタ領域の先頭アドレスは，この値を 14 ビット左シフトした “ 0x48d0000 ” となります。この際，.bpc セクション内の情報は次のようになります。

```
.bpc
  Address    00  01  02  03  04  05  06  07  -  08  09  0A  0B  0C  0D  0E  0F
0x00000000 : 00  00  8d  04                  -                                  ...
```

- 以上の処理は，リロケータブルなオブジェクト・ファイル作成時，実行可能オブジェクト・ファイル作成時を問わず行われます。
- .bpc セクションは，情報用の特殊な予約セクションであり，メモリにロードされることはありません。したがって，通常のセクションのように，リンク・ディレクティブに記述する必要はありません。

5.7.12 オプション・バイト

オプション・バイト機能を利用するにはアセンブリ言語ソースに次のような 6 バイトのデータを記述します。

```
        .section "OPTION_BYTES"
        .byte  0b00000001    -- 0x7a
        .byte  0b00000000    -- 0x7b
        .byte  0b00000000    -- 0x7c
        .byte  0b00000000    -- 0x7d
        .byte  0b00000000    -- 0x7e
        .byte  0b00000000    -- 0x7f
```

- オプション・バイト機能を持たないデバイスを指定した場合は，通常の入力セクションとして扱います。
- オプション・バイト機能を持つデバイスを指定し，本セクションの記述を省略した場合は，デバイス・ファイルに設定された初期値を使用します。
- 本セクションは必ず 6 バイト分を記述してください。6 バイト以下の場合は，次のメッセージを出力し，リンクを中止します。

```
F4112: illegal "section" section size.
```

- 設定不可能となっているビットは，初期値からの変更ができません。変更された場合は，次のメッセージを出力します。

```
W4613: illegal flash mask option access (file:"file" address:num1 bit:num2)
```

第 6 章　ROM 化プロセッサ

この章では，ROM 化プロセッサ（romp850）の概要，ROM 化の手順，操作方法などを説明します。

6.1　動作の流れ

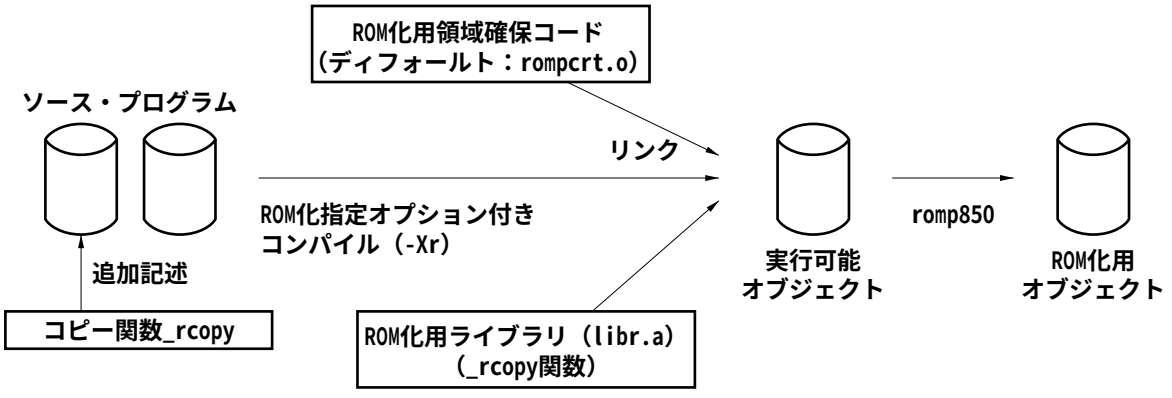
プログラム中で，グローバルに変数を宣言すると，初期値を持つ変数ならば data 属性のセクションへ，初期値を持たない変数ならば bss 属性のセクションというように，RAM 上のセクションに配置されます。特に初期値を持つ変数ならば，その初期値自体が RAM 上に配置されます。その他，アプリケーションの高速化のために，プログラム・コードを内蔵 RAM 領域へ配置する場合もあります。

組み込みシステムの場合，デバッグ時にインサーキット・エミュレータなどを使用する場合，実行可能なモジュールを配置イメージのままダウンロードして実行できます。しかし実際にプログラムをターゲット・システムの ROM 領域に書き込んで実行する場合，data 属性のセクションにある初期値情報や，RAM 領域に配置するプログラム・コードを，実行前に RAM 上に展開されていなければなりません。つまり RAM に展開するデータを ROM 上に持たせておき，それをアプリケーション実行前に ROM から RAM へコピーする作業が必要になります。

romp850 は，data 属性セクションの変数の初期値情報や，RAM 上に配置するプログラムを，1 つのセクションにパッキングするツールです。このセクションを ROM 上に配置し，CA850 で用意されているコピー関数を呼び出すことによって，初期値情報やプログラムを容易に RAM 上へ展開することができます。

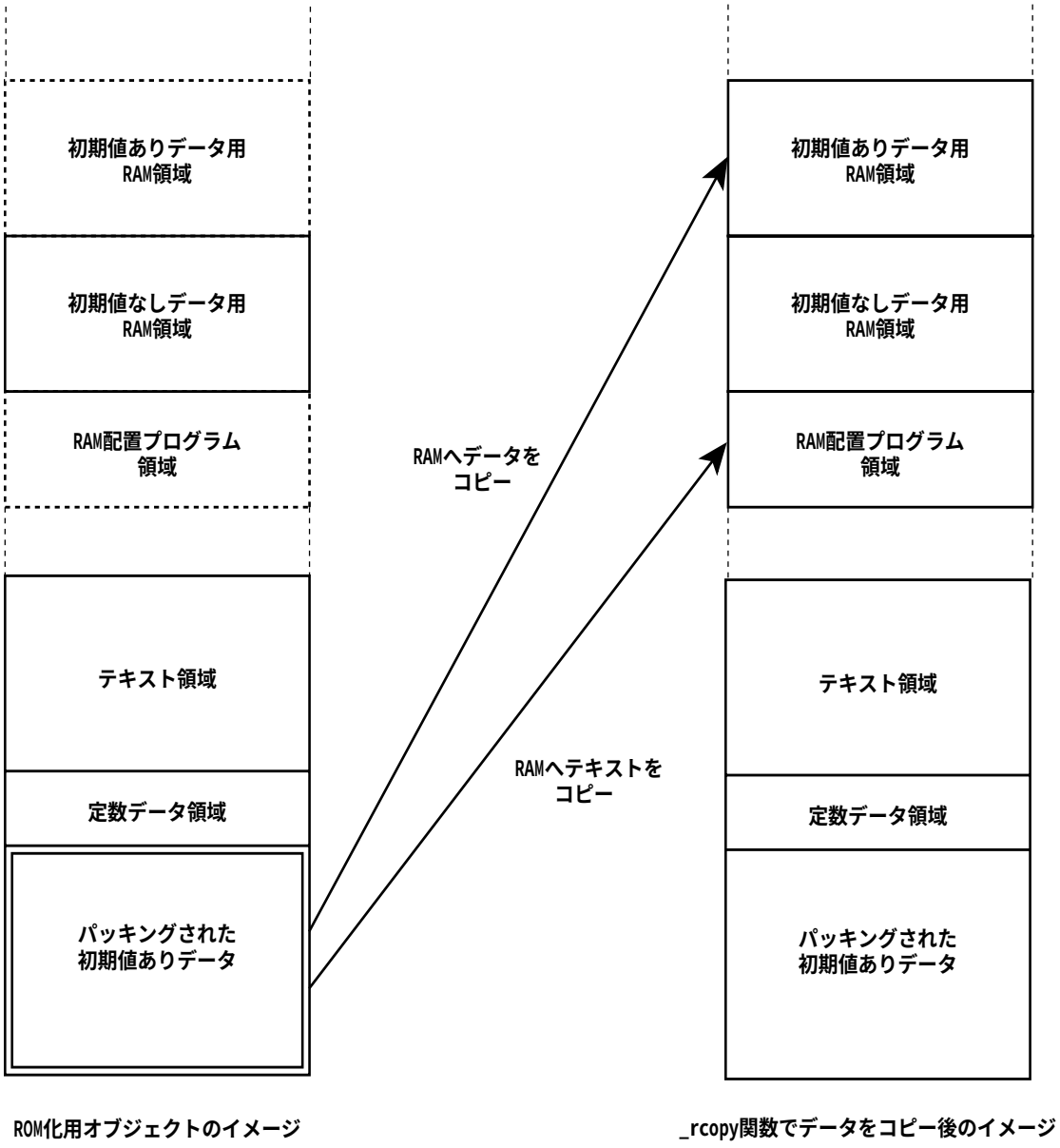
ROM 化用オブジェクトを作成する流れの概要は，次図のようになります。

図 6 － 1　ROM 化用オブジェクトの作成



図のように ROM 化用オブジェクトを作成すると，_rcopy 関数を実行することによって，RAM に配置するデータを，パッキングされた ROM からコピーします。イメージは次のようになります。

図 6 － 2 _rcopy 関数呼び出し前後のイメージ



ここで，ROM 化用オブジェクトに必要となるセクション名，およびそのセクションの先頭アドレス（ラベル名）は，デフォルトでは次のようになっています。

- パッキングしたセクション名→ rompsec セクション
- rompsec セクションの先頭アドレス（ラベル名）→ __S_romp

そして rompsec セクションから，RAM 領域へコピーする関数は次のとおりです。

- コピー関数→ _rcopy，_rcopy1，_rcopy2，_rcopy4 関数

この関数は lib850\r** にあるライブラリ “ libr.a ” に格納されています。

__S_romp は lib850\r** にある“ rompcrt.o ”で定義されているラベルです（このソース・ファイルは rompcrt.s）。rompcrt.o をそのまま使用することにより，romp850 によって自動的に .text 属性の直後（4 バイトでアラインしたところ）に，rompsec セクションを作成します。そして __S_romp が rompsec セクションの先頭アドレスを指すラベルになります。

このように自動的に rompsec セクションを作成する方法のほかに，rompcrt.s に相当するプログラムを独自に作成して配置することもできます。詳しくは，「6.4.2 作成手順（カスタマイズ）」を参照してください。

実際に ROM 化するには，この ROM 化用オブジェクトを作成してから，ヘキサ・ファイルに変換し，ROM 上に書き込むことになります。

なお，パッキングの必要なデータがアプリケーションに存在しなかった場合は，この ROM 化用オブジェクトを生成する必要はありません。ld850 で作成したオブジェクトを，そのままヘキサ・ファイルに変換してください。

また，romp850 は，リロケーション解決したオブジェクト・ファイルにシンボル情報，デバッグ情報が含まれる場合，それらを削除することなく ROM 化用のオブジェクト・ファイルを生成します。そのため，ROM 化後のオブジェクト・ファイルでもデバッガによるソース・デバッグができます。

6.2 入出力ファイル

romp850 では，次のファイルを入力ファイルとして扱うことができます。

- *file1.out* … ld850 によって出力された実行可能オブジェクト

出力されるファイルは，次のファイルです。

- *file2.out* … ROM 化用実行可能オブジェクト

入出力ファイル名は，ld850，romp850 にてそれぞれで指定できます。出力ファイルのデフォルトは romp.out です。

6.3 rompsec セクション

6.3.1 パッキングするセクションの種類

rompsec セクションとしてパッキングする対象となるものは，デフォルトでは「書き込み可能な属性を持つセクションに割り当てられたデータ」です。その他，オプション（-t オプション）を指定することによって「text 属性，const 属性を持つ任意のセクション」をパッキングすることも可能です。

具体的には次に示すようになります。

(1) 表 6 － 1 に示す予約セクション

(2) アセンブラ・プログラムにおいて .section 疑似命令により sdata 属性，または data 属性を指定し，任意の名前で生成したセクション，および内蔵命令 RAM に配置するセクション

表 6 － 1 romp850 によりパッキングされる予約セクション

.data,	.sdata,	.sedata,	.sidata,
.tidata,	.tidata.byte,	.tidata.word	

ただし，ユーザが指定する「text 属性，const 属性を持つ任意のセクション」をパッキングせず，さらに上記のセクションが実行可能モジュールに存在しない場合は，ROM 化オブジェクトを生成する必要はありません。

表 6 － 1 のセクションが存在するかしないかは，リンク・マップ・ファイルを参照してください。

なお，romp850 によって作成されたオブジェクト・ファイルをダンプ・コマンド（dump850）で参照することにより，.data セクションや .sdata セクションなどの代わりに rompsec セクションが作成されていることを確認できます。

6.3.2 rompsec セクションのサイズ

rompsec セクションとして確保されるサイズについて説明します。

ROM 化用モジュールを作成するときは，rompsec セクションのサイズと，使用している CPU の内蔵 ROM 領域，ターゲット・システムの ROM 領域のアドレスやサイズに注意します。rompsec セクションが，他のセクションとオーバラップしないようにリンク・ディレクティブ・ファイルを記述してください。具体的な記述例は，「6.4 ROM 化用オブジェクトの作成」を参照してください。

次に rompsec セクションのサイズを求める計算式を示します。

8 ＋ 16 ×（sdata ／ data 属性セクションの数）＋ sdata ／ data 属性セクションのサイズ
＋パディング・サイズ[注]

たとえば，.sdata，.data セクションが存在し，それぞれのサイズが 1002 バイト，1000 バイトで，それぞれのセクションの整列条件が 4 バイトの場合，rompsec セクションのサイズは次のようになります。

8 + 16 ×2 + 1002 + 1000 + 2 = 2044 バイト

注 ROM 化対象のセクションの整列条件により 1 セクションあたり 0 ～ 3 バイトになります。

6.3.3 rompsec セクションとリンク・ディレクティブ

ROM 化時には，.text セクションの直後に rompsec セクションが追加されます。よって，.text セクションを ROM の最後に配置することで，ROM の終端までの rompsec セクションを配置できます。

図 6 － 3 ROM 化処理を考慮したリンク・ディレクティブ

```
# 内蔵 ROM に SCONST ／ CONST ／ TEXT を配置
SCONST : !LOAD ?R {
       .sconst = $PROGBITS ?A .sconst;
};

CONST : !LOAD ?R {
       .const = $PROGBITS ?A .const;
};

# 内蔵 ROM の最後に .text を配置
TEXT : !LOAD ?RX {
       .pro_epi_runtime = $PROGBITS ?AX .pro_epi_runtime;
       .text = $PROGBITS ?AX .text;
};

# 外部 RAM に DATA を配置
DATA : !LOAD ?RX V0x100000 {
       .data = $PROGBITS ?AW;
       .sdata = $PROGBITS ?AWG;
       .sbss = $NOBIT ?AWG;
       .bss = $NOBIT ?AW;
};

# 内蔵 RAM に SIDATA を配置
SIDATA : !LOAD ?RX V0xffe000 {
       .sidata = $PROGBITS ?AW .sidata;
       .sibss = $NOBIT ?AWG .sibss;
};

__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;
```

rompsec セクションが内蔵 ROM 領域を越えた場合には，次のメッセージを出力して処理を中止します。

```
F8425: rompsec section overflowed highest address of target machine.
```

-rom_less オプションを指定することで，内蔵 ROM 領域を無視することができます。

また，-Ximem_overflow=warning オプションを指定することで，エラー・メッセージを警告メッセージにすることができます。

rompsec セクションを外部 ROM 領域の終端に配置する場合には，これらのチェックは行われません。メモリマップ情報を参照して，ROM に収まっているかの判断を行ってください。

なお，ROM の途中に rompsec セクションを配置する必要がある場合には，次のように rompsec セクションのサイズと配置アドレスから，rompsec セクションの配置される領域を認識した上で，rompsec セクション直後のセグメントに対して，適切なアドレス指定をしてください。

図 6 － 4 ROM 化処理を考慮したリンク・ディレクティブ（サイズ考慮）

```
# 内蔵 ROM に SCONST ／ CONST ／ TEXT を配置
SCONST : !LOAD ?R {
       .sconst = $PROGBITS ?A .sconst;
};

# 内蔵 ROM の途中に .text を配置
TEXT : !LOAD ?RX {
       .pro_epi_runtime = $PROGBITS ?AX .pro_epi_runtime;
       .text = $PROGBITS ?AX .text;
};

# TEXT と CONST の間に rompsec

# 内蔵 ROM の最後に rompsec のサイズを考慮したアドレス指定を行って CONST を配置
CONST : !LOAD ?R Vx3f800 {
       .const = $PROGBITS ?A .const;
};

# 外部 RAM に DATA を配置
DATA : !LOAD ?RX V0x100000 {
       .data = $PROGBITS ?AW;
       .sdata = $PROGBITS ?AWG;
       .sbss = $NOBIT ?AWG;
       .bss = $NOBIT ?AW;
};

# 内蔵 RAM に SIDATA を配置
SIDATA : !LOAD ?RX V0xffe000 {
       .sidata = $PROGBITS ?AW .sidata;
       .sibss = $NOBIT ?AWG .sibss;
};
__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;
```

6.4 ROM 化用オブジェクトの作成

6.4.1 作成手順（デフォルト）

ここでは，デフォルトで用意されている ROM 化用領域確保コード（rompcrt.o）を使用した方法を示します。

(1) アプリケーションの中で，コピー関数を呼び出します。

コピー関数はスタート・アップ・ルーチン内や main 関数の先頭など，なるべく始めの方で起動するようにします。コピー関数には _rcopy，_rcopy1，_rcopy2，_rcopy4 があり，それぞれ転送サイズに違いがあります（_rcopy と _rcopy1 は同じ）。これらについての詳細は「6.5 コピー関数」を参照してください。次図に例をあげます。

例では main 関数の先頭で _rcopy 関数を起動しています。

図 6 － 5 コピー関数 _rcopy の使用例 1

```
#define ALL_COPY(-1)

int _rcopy(unsigned long *, long);
extern unsigned long _S_romp;

void main(void)
{
        int     ret;

        ret = _rcopy(&_S_romp, ALL_COPY);

              :
}
```

(2) ROM 化時には，.text セクションの直後に rompsec セクションが追加されます。

.text セクションを ROM の最後に配置することで，ROM の終端までの rompsec セクションを配置できます（「図 6 － 3 ROM 化処理を考慮したリンク・ディレクティブ」参照）。

(3) コンパイル・オプションで “ ROM 化用オブジェクトの生成 ” を指定します。

- コマンド・ラインからの場合

  コンパイル・オプション “ -Xr ” を追加指定します。

- PM+ からの場合

  [コンパイラ共通オプションの設定] ダイアログの [ROM 化] タブにある “ ROM 化用オブジェクトの生成 ” をチェックします。

これにより，__S_romp というラベルが，オブジェクト内の .text セクションの終端を越える最初のアドレスを指すコードが生成されます。

(4) コンパイル，リンクを行います。

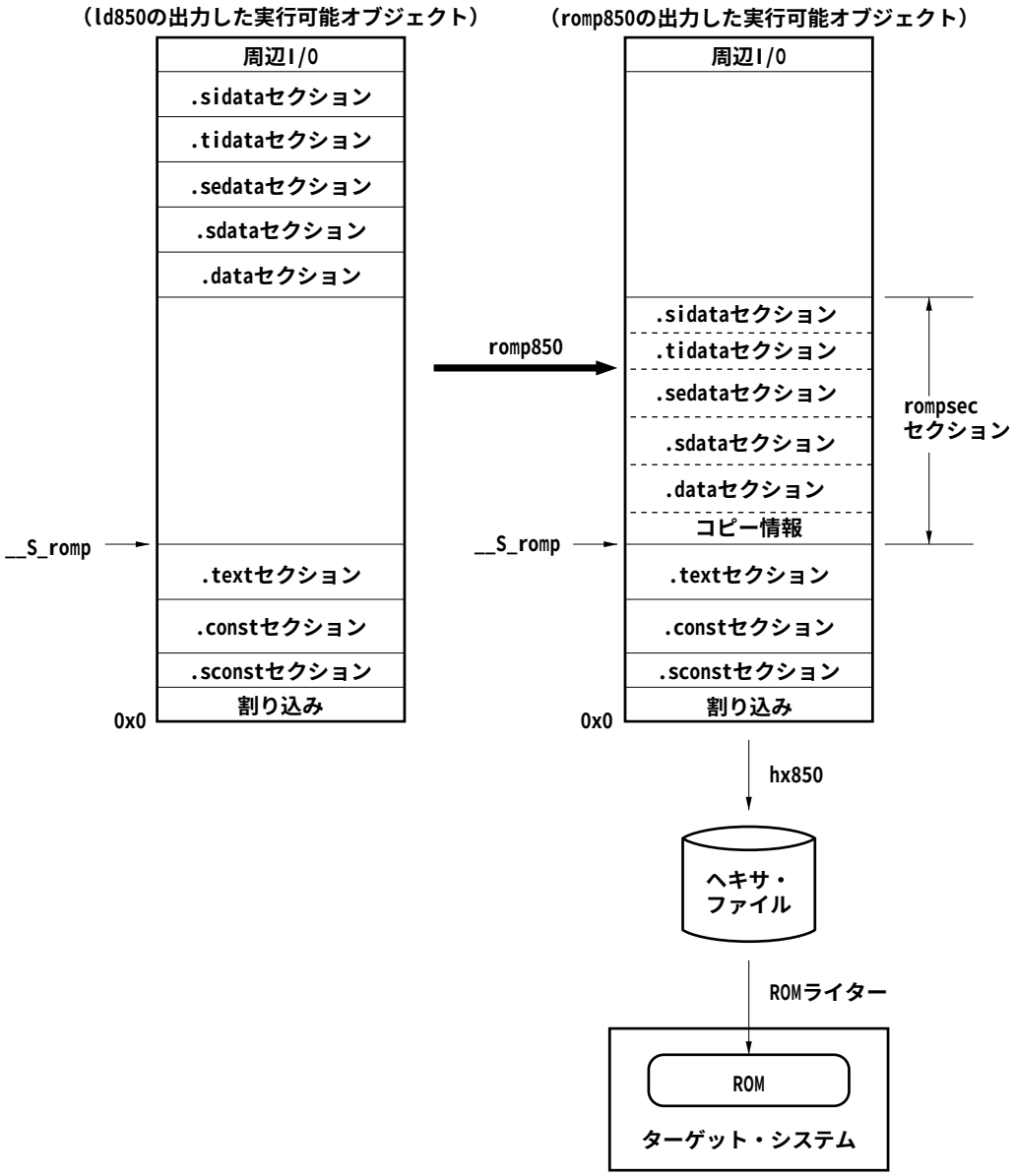
ca850 対して「ROM 化用オブジェクトの生成」を指示することにより，ROM 化用領域確保コードである“ rompcrt.o ”（lib850\r** 内に存在）と，_rcopy 関数が格納されている“ libr.a ”が自動的にリンクされます。この際，リンクする順番が関係します。“ rompcrt.o ”は，TEXT 属性群の最後にリンクする必要がありますので，コマンド・ラインから起動している場合，-l オプションでリンク指定するライブラリ群よりも後にリンクしてください。PM+ を使用している場合は，自動的に TEXT 属性群の最後にリンクするため，特に意識する必要はありません。

(5) ROM 化プロセッサ（romp850）を起動します。

(4) で完成した実行可能モジュールから，romp850 を使用して ROM 化用モジュールを生成します。

なお，PM+ で“ ROM 化用オブジェクトの生成 ”を指定している場合は，(4) から (5) まで自動的に行われ，ヘキサ・ファイルまで生成されます。コマンド・ラインから起動している場合は，ca850 から ld850 の起動まで行って実行可能モジュールを生成したあと，romp850 を起動して ROM 化用オブジェクトを生成します。マップのイメージを図にすると，次のようになります。

図 6 － 6 ROM 化のイメージ 1



6.4.2 作成手順（カスタマイズ）

ここでは，ROM 化用領域確保コードにあたる “ rompcrt.o ” を独自に作成し，rompcrt セクションの先頭アドレスや配置場所を自分で決定する方法を示します。

(1) デフォルトの ROM 化用領域確保コード “ rompcrt.s ” に相当するコードを記述します。

ここではファイル名を “ rompack.s ”，ROM 化用領域の先頭を指すシンボル名を “ __rompack ” とします。また，このとき，このシンボルの存在するセクションを“ rompack セクション ”とします。この場合，rompack.s は，次のようなコードになります。

図 6 － 7 rompack.s の例

```
        .file  "rompack.s"
        .section".rompack",text
        .align 4
        .globl __rompack, 4
__rompack:
```

(2) アプリケーションの中で，コピー関数を呼び出します。

コピー関数はスタート・アップ・ルーチン内や main 関数の先頭など，なるべく始めの方で起動するようにします。コピー関数には _rcopy，_rcopy1，_rcopy2，_rcopy4 があり，それぞれ転送サイズに違いがあります（_rcopy と _rcopy1 は同じ）。これらについての詳細は「6.5 コピー関数」を参照してください。次図に例をあげます。例では main 関数の先頭で _rcopy 関数を起動しています。

図 6 － 8 コピー関数 _rcopy の使用例 2

```
#define ALL_COPY (-1)

int _rcopy(unsigned long *, long);
extern unsigned long _rompack;

void main(void)
{
        int    ret;

        ret = _rcopy(&_rompack, ALL_COPY);
              :
}
```

(3) リンク・ディレクティブで，作成した rompack セクションを定義します。

これと同時にアドレスを指定すると，rompack セクションの配置場所を任意に決定することもできます。例として rompack セクションを含むセグメントを ROMPACK とし，このセグメントを 0x3000 番地に配置するとした場合，リンク・ディレクティブは次のようになります。

図 6 － 9 リンク・ディレクティブの指定例

```
TEXT:  !LOAD ?RX V0x1000 {
       .text = $PROGBITS ?AX .text;
};

ROMPACK: !LOAD ?RX V0x3000 {
       .rompack = $PROGBITS ?AX .rompack;
};
              :
```

このとき，ROMPACK セグメントの配置アドレスが，前後のセグメントと重ならないように，rompack セクションのサイズを，「6.3.2 rompsec セクションのサイズ」にしたがって見積もり，リンク・ディレクティブ・ファイルに反映します。

(4) コンパイル・オプションで “ ROM 化用オブジェクトの生成 ” を指定します。

- コマンド・ラインからの場合

  コンパイル・オプション “ -Xr ” を追加指定します
- PM+ からの場合

  [コンパイラ共通オプションの設定] ダイアログの [ROM 化] タブにある“ ROM 化用オブジェクトの生成 ” をチェックします。

これにより，ラベル “ rompack ” が rompsec と同じアドレスを指すコードを生成します。

(5) コンパイラ共通オプション，ROM 化プロセッサ・オプションを指定します。

- コマンド・ラインからの場合

  ROM 化プロセッサのオプションで，ROM 化用領域確保コードのエントリ・シンボルを指定する “ -b オプション ” で “ __rompack ” を指定します。
- PM+ からの場合

  [コンパイラ共通オプションの設定] ダイアログの“ 入力ファイル ”で“ rompcrt ファイル : ”に“ rompack.s ”, または “ rompack.o ” を追加します。[ROM 化プロセッサオプションの設定] ダイアログの [オプション] タブで “ エントリラベル [-b] ” に rompack セクションの先頭ラベルである “ __rompack ” を記述します。

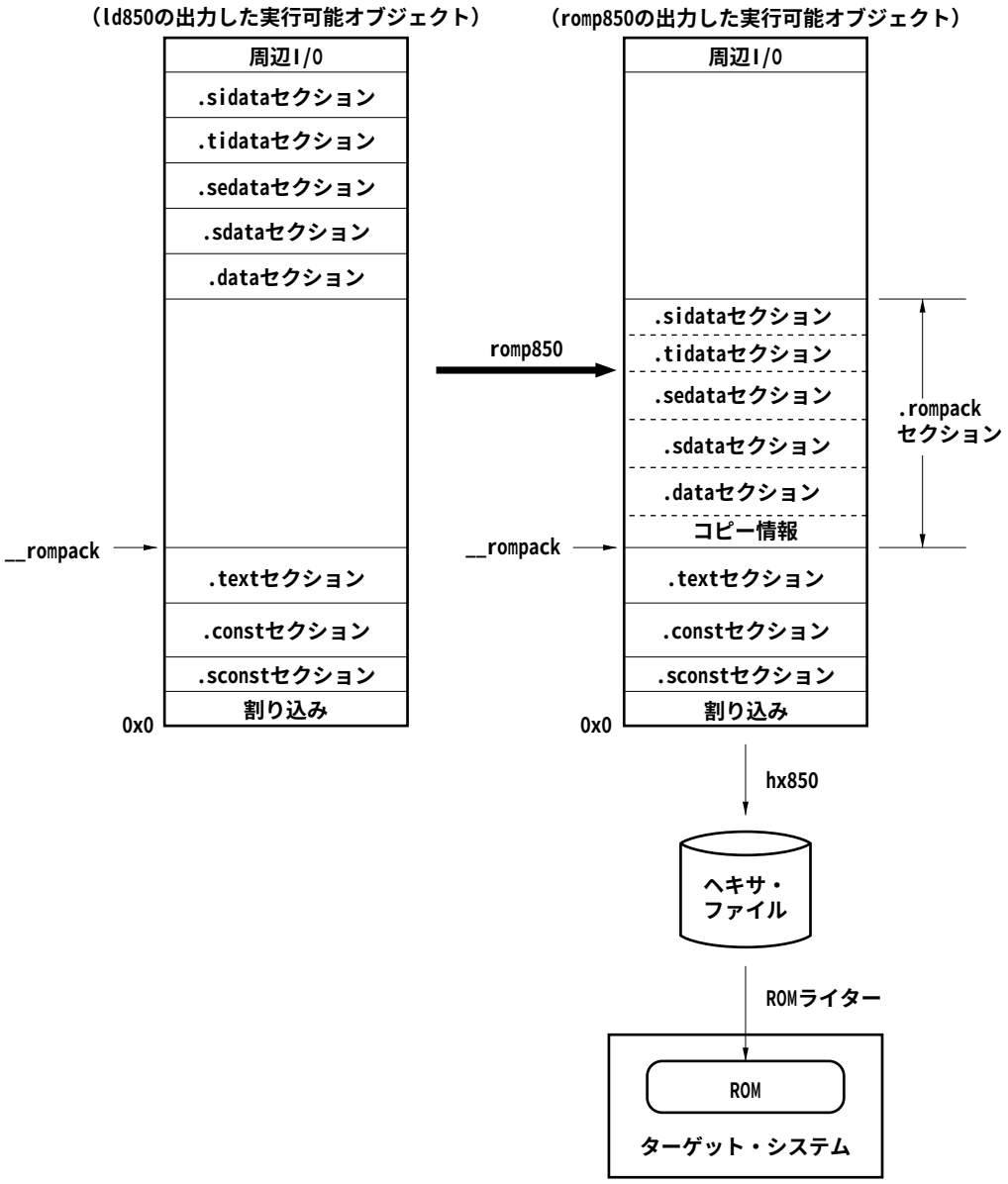
(6) コンパイル，リンクを行います。

ca850 に対して「ROM 化用オブジェクトの生成」を指示することにより，_rcopy 関数が格納されている “ libr.a ” が自動的にリンクされます。

(7) ROM 化プロセッサ（romp850）を起動する。

(6) で完成した実行可能モジュールから，romp850 を使用して ROM 化用モジュールを作成します。なお，PM+ を使用し，“ ROM 化用オブジェクトの生成 ” を指定している場合は，(6) から (7) まで自動的に行われ，ヘキサ・ファイルまで作成します。コマンド・ラインから起動している場合は，ca850 から ld850 の起動まで行って実行可能モジュールを作成し，romp850 を起動して ROM 化用オブジェクトを作成します。マップのイメージを次図に示します。

図 6 － 10 ROM 化のイメージ 2



6.5 コピー関数

6.5.1 コピー・ルーチン

ここでは，ROM 化するプログラムに必要なコピー・ルーチン（_rcopy）について説明します。

表 6 － 2 コピー・ルーチン一覧

関数名	機能
_rcopy	ROM 化セクションのコピー（1 バイト転送）
_rcopy1	ROM 化セクションのコピー（1 バイト転送）
_rcopy2	ROM 化セクションのコピー（2 バイト転送）
_rcopy4	ROM 化セクションのコピー（4 バイト転送）

転送先の RAM の仕様に応じて，1 バイト転送，2 バイト転送，4 バイト転送を使い分けてください。各関数の仕様は次のようになります。

_rcopy

〔概要〕

_rcopy

初期値データ／RAM テキスト[注]のコピー（1 バイト）

注　RAM に配置する初期値ありデータ・セクション，および内蔵 RAM 用テキスト・セクションです。

〔形式〕

```
int                 _rcopy(&label, number)
unsigned long       label;
long                number;
```

〔説明〕

_rcopy(&label, number) は，label の示すアドレス以降に存在する rompsec セクション内の情報を元に，コピーしたいセクション番号 number の初期値データ，または RAM に配置するテキストを，RAM 領域に 1 バイトずつコピーします。number に -1 を指定した場合，rompsec セクション内のすべてのセクションをコピーします。セクション番号 number は，1 から始まる正数です。

デフォルトは，セクションが入力ファイル中に出現した順番に割り当てられます。romp850 のオプション“ -p オプション ”,“ -t オプション ”で rompsec セクションに配置するセクションを指定した場合は，指定した順番に割り当てられます。

ただし，PM+ において“ ROM 化セクションファイル ”を作成すると，#define による“ 番号 ”と“ ラベル ”の対応付けされた C 言語ソース・ヘッダファイルが生成され，ラベル名によって，number を指定することができます。

具体的な使用例は「6.5.2 使用例」を参照してください。

〔戻り値〕

0	正常終了（正しくコピーされた場合）
-1	異常終了（正しくコピーされなかった場合）

〔注意事項〕

- label の示すアドレスが rompsec セクションの先頭でなかった場合はコピーを行いません。
- _rcopy は romp850 で生成された情報にしたがってコピーを行います。
  _rcopy 実行時にコピー先のアドレスにオフセットを加えるような処理はできません。
- コピーを行うとオーバライトが生じる場合，コピーを行いません。
- _rcopy の第一引数 label には，絶対値を持つグローバルなラベル，または絶対アドレスを指定してください。これら以外のものを指定した場合，その結果は保証されません。
- _rcopy 関数は _rcopy1 関数と同じ機能です。旧版からの互換性のために _rcopy を用意してあります。

_rcopy1

〔概要〕

_rcopy1

初期値データ／RAM テキスト[注]のコピー（1 バイト）

注　RAM に配置する初期値ありデータ・セクション，および内蔵 RAM 用テキスト・セクションです。

〔形式〕

```
int                 _rcopy1(&label, number)
unsigned long       label;
long                number;
```

〔説明〕

_rcopy1(&label, number) **は，**label **の示すアドレス以降に存在する** rompsec **セクション内の情報を元に，コピーしたいセクション番号** number **の初期値データ，または** RAM **に配置するテキストを，**RAM **領域に** 1 **バイトずつコピーします。**number **に** -1 **を指定した場合，**rompsec **セクション内のすべてのセクションをコピーします。セクション番号** number **は，**1 **から始まる正数です。**

デフォルトは，セクションが入力ファイル中に出現した順番に割り当てられます。romp850 **のオプション“** -p **オプション ”,“** -t **オプション ”で** rompsec **セクションに配置するセクションを指定した場合は，指定した順番に割り当てられます。**

ただし，PM+ **において“** ROM **化セクションファイル ”を作成すると，**#define **による“ 番号 ”と“ ラベル ”の対応付けされた** C **言語ソース・ヘッダファイルが生成され，ラベル名によって，**number **を指定することができます。**

具体的な使用例は「6.5.2 使用例**」を参照してください。**

〔戻り値〕

0	**正常終了（正しくコピーされた場合）**
-1	**異常終了（正しくコピーされなかった場合）**

〔注意事項〕

- label **の示すアドレスが** rompsec **セクションの先頭でなかった場合はコピーを行いません。**
- _rcopy1 **は** romp850 **で生成された情報にしたがってコピーを行います。**
  _rcopy1 **実行時にコピー先のアドレスにオフセットを加えるような処理はできません。**
- **コピーを行うとオーバライトが生じる場合，コピーを行いません。**
- _rcopy1 **の第一引数** label **には，絶対値を持つグローバルなラベル，または絶対アドレスを指定してください。これら以外のものを指定した場合，その結果は保証されません。**
- _rcopy1 **関数は** _rcopy **関数と同じ機能です。旧版からの互換性のために** _rcopy **を用意してあります。**

_rcopy2

〔概要〕

_rcopy2

初期値データ／RAM テキスト[注]のコピー（2 バイト）

注　RAM に配置する初期値ありデータ・セクション，および内蔵 RAM 用テキスト・セクションです。

〔形式〕

```
int                 _rcopy2(&label, number)
unsigned long       label;
long                number;
```

〔説明〕

_rcopy2(&label, number) **は，**label **の示すアドレス以降に存在する** rompsec **セクション内の情報を元に，コピーしたいセクション番号** number **の初期値データ，または** RAM **に配置するテキストを，**RAM **領域に** 2 **バイトずつコピーします。**number **に** -1 **を指定した場合，**rompsec **セクション内のすべてのセクションをコピーします。セクション番号** number **は，**1 **から始まる正数です。**

デフォルトは，セクションが入力ファイル中に出現した順番に割り当てられます。romp850 **のオプション“** -p **オプション ”,“** -t **オプション ”で** rompsec **セクションに配置するセクションを指定した場合は，指定した順番に割り当てられます。**

ただし，PM+ **において“** ROM **化セクションファイル ”を作成すると，**#define **による“ 番号 ”と“ ラベル ”の対応付けされた** C **言語ソース・ヘッダファイルが生成され，ラベル名によって，**number **を指定することができます。**

具体的な使用例は「6.5.2 使用例**」を参照してください。**

〔戻り値〕

0	**正常終了（正しくコピーされた場合）**
-1	**異常終了（正しくコピーされなかった場合）**

〔注意事項〕

- label **の示すアドレスが** rompsec **セクションの先頭でなかった場合はコピーを行いません。**
- _rcopy2 **は** romp850 **で生成された情報にしたがってコピーを行います。**
  _rcopy2 **実行時にコピー先のアドレスにオフセットを加えるような処理はできません。**
- **コピーを行うとオーバライトが生じる場合，コピーを行いません。**
- _rcopy2 **の第一引数** label **には，絶対値を持つグローバルなラベル，または絶対アドレスを指定してください。これら以外のものを指定した場合，その結果は保証されません。**

_rcopy4

〔概要〕

_rcopy4

初期値データ／RAM テキスト[注]のコピー（4 バイト）

注　RAM に配置する初期値ありデータ・セクション，および内蔵 RAM 用テキスト・セクションです。

〔形式〕

```
int                 _rcopy4(&label, number)
unsigned long       label;
long                number;
```

〔説明〕

_rcopy4(&label, number) は，label の示すアドレス以降に存在する rompsec セクション内の情報を元に，コピーしたいセクション番号 number の初期値データ，または RAM に配置するテキストを，RAM 領域に 4 バイトずつコピーします。number に -1 を指定した場合，rompsec セクション内のすべてのセクションをコピーします。セクション番号 number は，1 から始まる正数です。

デフォルトは，セクションが入力ファイル中に出現した順番に割り当てられます。romp850 のオプション“ -p オプション ”,“ -t オプション ”で rompsec セクションに配置するセクションを指定した場合は，指定した順番に割り当てられます。

ただし，PM+ において“ ROM 化セクションファイル ”を作成すると，#define による“ 番号 ”と“ ラベル ”の対応付けされた C 言語ソース・ヘッダファイルが生成され，ラベル名によって，number を指定することができます。

具体的な使用例は「6.5.2 使用例」を参照してください。

〔戻り値〕

0	正常終了（正しくコピーされた場合）
-1	異常終了（正しくコピーされなかった場合）

〔注意事項〕

- label の示すアドレスが rompsec セクションの先頭でなかった場合はコピーを行いません。
- _rcopy4 は romp850 で生成された情報にしたがってコピーを行います。
  _rcopy4 実行時にコピー先のアドレスにオフセットを加えるような処理はできません。
- コピーを行うとオーバライトが生じる場合，コピーを行いません。
- _rcopy4 の第一引数 label には，絶対値を持つグローバルなラベル，または絶対アドレスを指定してください。これら以外のものを指定した場合，その結果は保証されません。

6.5.2 使用例

(1) すべてのセクションを 1 バイト転送する場合

```
extern unsigned long _S_romp;

main()
{
       int    ret;

       ret = _rcopy(&_S_romp, -1);
       /* -Xr 指定により，絶対値を持つグローバルなラベルを指定 */
}
```

このように，ca850 に ROM 化用オプションを指定することにより，ラベルは絶対アドレス参照となります。したがって，_rcopy() をアセンブリ言語ソース・プログラムで呼び出す場合は，次のようにしてください。

```
       .extern __S_romp, 4        -- 外部ラベルとして宣言

       mov    #__S_romp, r6       -- __S_romp の絶対アドレスを第一
       mov    -1, r7              -- 引数，-1 を第二引数として
       jarl   __rcopy, lp         -- _rcopy を呼び出し
```

(2) 1 ～ 6 のセクションを 4 バイト転送，7 ～ 11 のセクションを 1 バイト転送する場合

```
extern unsigned long _S_romp;

main()
{
       int    ret, num;

       for(num = 1; num<=6; num++) {
              ret = _rcopy4(&_S_romp, num);
              if(ret == -1) {
                     /* エラー処理 */
              }
       }

       for(num = 7; num <= 11; num++) {
              ret = _rcopy1(&_S_romp, num);
              if(ret == -1) {
                     /* エラー処理 */
              }
       }
}
```

(3) 誤った指定の例 1

```
extern unsigned long _S_romp;
char *cp;

func()
{
        int    ret;

        cp = &_S_romp;        /* 変数に入れたため，第一引数が gp 相対値となる */
        ret = _rcopy(cp, -1);
}
```

(4) 誤った指定の例 2

```
extern unsigned long _S_romp;
int    i;

func()
{
        int    ret;

        i = 0x100;            /* 変数に入れたため，第一引数が gp 相対値となる */
        ret = _rcopy(i, -1);
}
```

- number に指定するセクション番号は 1 から始まる正の整数です。
  セクション名とセクション番号の関連は，メモリ・マップから参照できますが，PM+ を使用した場合“ROM 化セクションファイル”の出力により，セクション番号とラベルの対応付けがされた C 言語ヘッダ・ファイルを作成できます。つまり，number にラベルを使用することができます。ROM 化セクションファイルの出力方法，ラベル名の規則については「6.8.1 [ROM 化プロセッサオプションの設定] ダイアログ」を参照してください。
- number にセクション番号，または -1 以外の指定を行った場合，コピーを行いません。
- 複数の RAM が存在し，複数のコピー・ルーチンを使いわける場合，number に -1 の指定を行うと，すべてのセクション整列等の問題により，正常にコピーされません。
  number には -1 の指定をせず，セクション番号を指定してください。
- number に -1 を指定した場合，セクション番号順にコピーを行います。
  途中で上記の各問題によりコピーが行われないセクションが発生した場合，戻り値として -1 を返却します。問題となったセクションよりも後のセクションは，コピーされません。

6.6 操作方法

この節では，romp850 の操作方法について説明します。

6.6.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
romp850 [ オプション ] ... ファイル名
        [ ]    ：[ ] 内は省略できます。
        ...    ：直前の [ ] 内のパターンの繰り返しができます。
```

6.6.2 PM+ による方法

[ROM 化プロセッサオプションの設定] ダイアログは，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [ROM 化プロセッサオプションの設定] を選択

romp850 の起動は，プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

なお，ROM 化プロセッサが出力する実行可能オブジェクト・ファイル名は，[プロジェクトの設定] ダイアログで [ソース・ファイル] タブ選択時の “ ソース・ファイル ” に表示される最初のファイルの拡張子を除いた名前に，“ .out ” を付加したものとなり，リンカが出力するオブジェクト・ファイル名と同じです。デフォルトは romp.out で，[ファイル] タブの “ 出力ファイル [-o] ” で指定できます。

romp850 は，[コンパイラ共通オプションの設定] ダイアログの [ROM 化] タブの “ ROM 化用オブジェクトの生成 ” を指定した場合にのみ，PM+ から起動されます。

6.7 オプションの種類と機能

この節では，romp850 **のオプションを示します。**

〔オプション説明でのマーク〕

【PM+】	PM+ **で指定項目が存在するオプション**

6.7.1 ファイル

ファイルの出力ファイル・オプションを設定します。

+err_file=*file*

エラー・メッセージをファイル *file* に追加保存します。

-err_file=*file*

エラー・メッセージをファイル *file* に上書き保存します。

-o *ofile*

【PM+】

生成されるオブジェクト・ファイル名を *ofile* とします。このオプションを省略した場合，romp.out が指定されたものとみなします。ファイル名として a.out を指定することはできません。

6.7.2 オプション

romp850 の通常のオプションを設定します。

-Ximem_overflow=warning

【PM+】

内蔵 ROM ／ RAM をオーバーフローした際のチェックの制御を行います。オーバーフロー時には警告メッセージを出力し，処理を継続します。このオプションを省略した場合，オーバーフロー時にはエラー・メッセージを出力し，処理を中止します。

-b *label*

【PM+】

ラベル *label* 値を，生成される rompsec セクションの先頭アドレスとします。指定したラベルがオブジェクト・ファイル中に存在しない場合，またはこのオプションを複数回指定した場合，メッセージを出力し処理を中止します。このオプションを省略した場合，__S_romp が指定されたものとみなします。

-d

【PM+】

生成するファイル中に text 属性を持つセクションを入れずに，rompsec セクションのみを持つオブジェクト・ファイルを生成します。このオプションを省略した場合，text 属性を持つセクションも入れます。

-i

【PM+】

入力ファイル，および出力ファイルのアドレスの重複チェックを行いません。

-m[=*mapfile*]

【PM+】

生成するオブジェクト・ファイルのメモリ・マップを *mapfile* に出力します。*mapfile* を省略した場合，標準出力に出力します。

-p *section*

【PM+】

セクション名 *section* の内容とそのアドレス，およびサイズの情報を rompsec セクションに入れます。このオプションは，data 属性，または sdata 属性を持つセクションに関するオプションです。また，複数回指定した場合，指定した順に rompsec セクションに入れます。指定したセクションがオブジェクト・ファイル中に存在しない場合，メッセージを出力し，処理を中止します。このオプションを省略した場合，data 属性，または sdata 属性を持つすべてのセクション，および内蔵命令 RAM に配置されるセクションが指定されたものとみなします。なお，セクション名に空白は使用できません。

-rom_less

【PM+】

rompsec セクションに対して，内蔵 ROM 周辺の配置エラー・チェックを行いません。ROM レス・モード使用時に指定することを推奨します。また，シングルチップ・モード選択時の内蔵 ROM オーバのチェックには対応していません。このオプションを指定して内蔵 ROM オーバーフローのチェックを無効とし，dump850 で確認してください。このオプションを省略した場合，rompsec セクションに対して，内蔵 ROM 周辺の配置エラー・チェックを行います。

-t *section*

【PM+】

セクション名 *section* の内容とそのアドレス，およびサイズの情報を rompsec **セクションに入れます。**

このオプションは text **属性，または** const **属性を持つセクションに関するオプションです。また，複数回指定した場合，指定した順に** rompsec **セクションに入れます。指定したセクションがオブジェクト・ファイル中に存在しない場合，メッセージを出力し，処理を中止します。このオプションで指定できるセクションは，**text **属性，または** const **属性を持つセクションで，これ以外の属性のセクションを指定した場合は，メッセージを出力し，処理を中止します。なお，セクション名に空白は使用できません。**

このオプションを省略した場合,内蔵命令RAM**に配置される各セクションが指定されたものとみなします。**

また，内蔵命令 RAM **搭載のデバイス・ファイルを指定してリンクされた入力ファイルに対して，このオプションで特定のセクションを指定した場合，指定されなかった内蔵命令** RAM **に配置されたセクションは，**rompsec **セクションに入らないだけでなく，出力ファイル中からも削除されます。**

6.7.3 その他

その他のオプションを設定します。

-F *devpath*

デバイス・ファイルをフォルダ *devpath* で探します。このオプションを省略した場合，標準フォルダで探します。

-V

romp850 のバージョン情報を標準エラー出力に出力し，終了します。

-help

romp850 のオプションの説明を標準エラー出力に出力します。

@cfile

cfile をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

6.8 PM+ での設定

この節では，対象プロジェクトのソース・ファイルに対して，romp850 のコマンド・オプションを設定するダイアログについて説明します。

6.8.1 [ROM 化プロセッサオプションの設定] ダイアログ

[ROM 化プロセッサオプションの設定] ダイアログの上部には，次の 4 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

表 6 － 3 [ROM 化プロセッサオプションの設定] ダイアログ

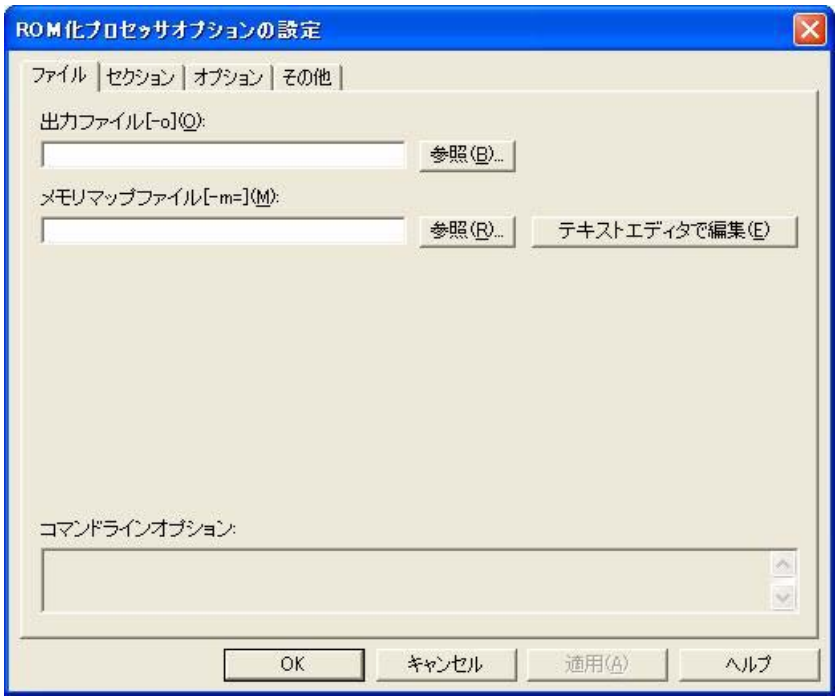
タブ	内容
[ファイル]	ファイルの設定
[セクション]	ROM 化するセクションの設定
[オプション]	オプションの設定
[その他]	その他の設定

なお，[ROM 化プロセッサオプションの設定] ダイアログには，コマンド・ラインから起動する場合のオプション名が “ [ ] ” に表示されています。

[ファイル]

ROM 化プロセッサのファイルの設定を行います。

図 6 － 11 [ROM 化プロセッサオプションの設定] ダイアログ：[ファイル] タブ



(1) 出力ファイル [-o]

出力ファイル名を設定するエディット・ボックスです。ファイル名には空白を指定することができません。出力ファイル名を省略した場合，出力ファイルは romp.out になります。

また，[参照] ボタンにより表示されるダイアログで，ファイルを選択することができます。

(2) メモリマップファイル [-m=]

romp850 を起動した結果のマッピング情報を出力するファイル名を設定するエディット・ボックスです。

[テキストエディタで編集] ボタンで，指定したファイルを開くことができます（編集することができます）。

(3) コマンドラインオプション

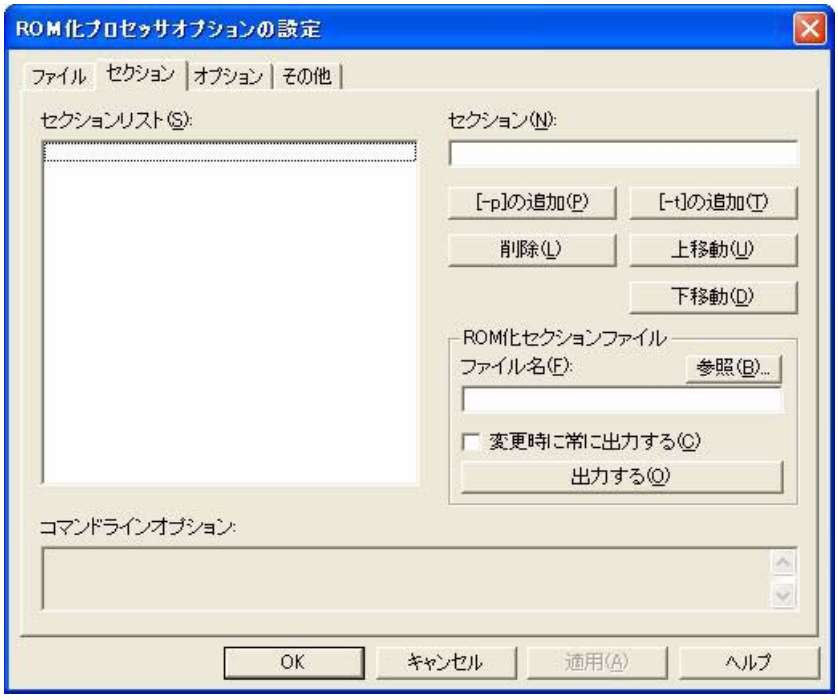
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[セクション]

ROM 化するセクションの設定を行います。

図 6 － 12 [ROM 化プロセッサオプションの設定] ダイアログ：[セクション] タブ



(1) セクションリスト

ROM 化するセクションとオプションを指定順に並べたリスト・ボックスです。ここで指定する順番が，rompsec セクションに格納される順番になります。つまり，_rcopy，_rcopy1，_rcopy2，_rcopy4 の第二引数で指定する番号に相当します（1 から順番に振られます）。

ただし，(3) で説明する “ ROM 化セクションファイル ” を作成すると，#define による “ 番号 ” と “ ラベル ” の対応付けされた C 言語ソース・ヘッダファイルが生成され，ラベル名によって，第二引数で指定する番号を指定できます。

(2) セクション

リストのセクションを追加する場合はセクション名を記述して [[-p] の追加] ボタン，または [[-t] の追加] ボタンで追加します。

(a) [-p] の追加

“ セクションリスト ” に -p オプションとして追加します。“ セクション ” に指定したセクションを，-p オプションで指定するセクションとして “ セクションリスト ” に追加します。つまり，“ セクション ” に指定したセクションが data 属性，sdata 属性だった場合は，このボタンを使ってセクションリストに追加します。

(b) [-t] の追加

“ セクションリスト ” に -t オプションとして追加します。“ セクション ” に指定したセクションを，-t オプションで指定するセクションとして “ セクションリスト ” に追加します。つまり，“ セクション ” に指定したセクションが text 属性，const 属性だった場合は，このボタンを使ってセクションリストに追加します。

(c) 削除

“ セクションリスト ” の現在の選択位置のセクションを削除します。

(d) 上移動

“ セクションリスト ” の現在の選択位置のセクションを 1 つ上に移動します。

(e) 下移動

“ セクションリスト ” の現在の選択位置のセクションを 1 つ下に移動します。

(3) ROM 化セクションファイル

ROM 化セクション・ファイルとは，_rcopy，_rcopy1，_rcopy2，_rcopy4 の第二引数で指定する番号を，アプリケーション中で指定しやすくするためのファイルで，#define による “ 番号 ” と “ ラベル ” の対応付けした C 言語ソース・ヘッダ・ファイルです。つまり，第二引数にラベルを指定することで，rompsec セクションからコピーしたいセクションを指定することができます。たとえば，.text，.data，.const，text1 をセクションリストに登録し，ROM 化セクションファイルを出力すると，

```
#define ROMPSCN__text 1
#define ROMPSCN__data 2
#define ROMPSCN__const 3
#define ROMPSCN_text1 4
```

と出力されます。

(a) ファイル名

ファイル名が指定されている場合は (b)，または (c) のときにファイルを出力します。

(b) 変更時に常に出力する

チェックされている場合は，セクションを変更して [OK] ボタン，または [適用] ボタンを押したときにファイルを出力します。

(c) 出力する

[出力する] ボタンを押したときにファイルを出力します。

なお，ファイルを正常に出力した場合と失敗した場合はメッセージ・ボックスが表示されます。

ROM 化セクション・ファイルは次の #define がセクションの数だけ出力されます。

#define ROMPSCN_< セクション名 > < セクション番号 >

プリプロセッサの識別子として使用できない文字はすべて “ _ ” に変換され，同じ識別子が存在する場合はメッセージ・ボックスが表示されてファイルは出力されません。

(4) コマンドラインオプション

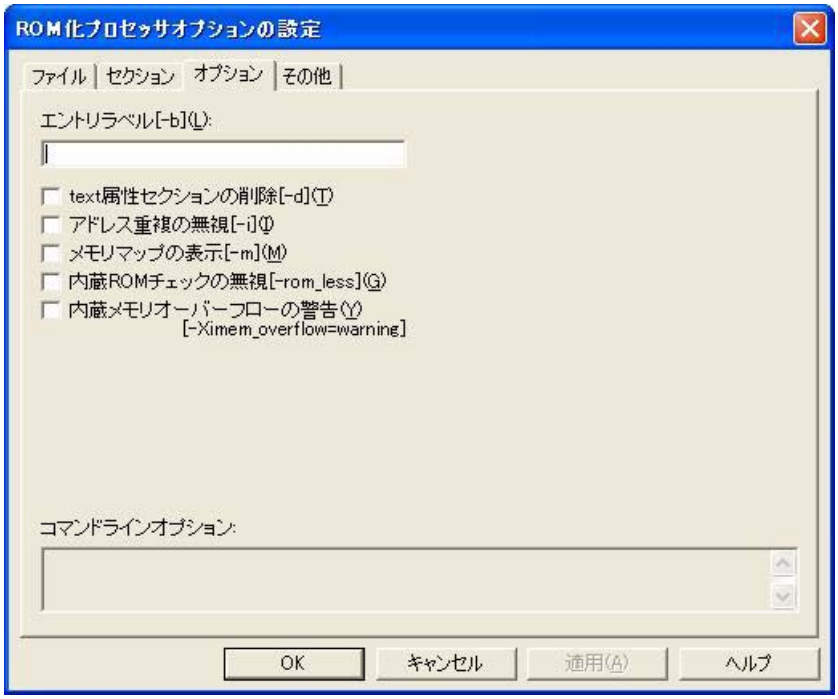
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[オプション]

ROM 化プロセッサのオプションの設定を行います。

図 6 － 13 [ROM 化プロセッサオプションの設定] ダイアログ：[オプション] タブ



(1) エントリラベル [-b]

指定したラベルの値を，生成される rompsec セクションの先頭アドレスとします。指定したラベルがオブジェクト・ファイル中に存在しない場合，メッセージを出力し処理を中止します。このオプションを省略した場合，__S_romp が指定されたものとみなします。

(2) text 属性セクションの削除 [-d]

生成するファイル中に text 属性を持つセクションを含めずに，rompsec セクションのみを持つオブジェクト・ファイルを生成します。このオプションを省略した場合，text 属性を持つセクションも含めます。

(3) アドレス重複の無視 [-i]

入力ファイル，および出力ファイルのアドレスの重複チェックを行いません。

(4) メモリマップの表示 [-m]

romp850 が生成するオブジェクト・ファイルのメモリ・マップを PM+ のアウトプット・ウインドウに出力します。

なお，メモリ・マップは，プロジェクト・フォルダに自動生成されるログ・ファイル（プロジェクト名 + .plg）に含まれます。

(5) 内蔵 ROM チェックの無視 [-rom_less]

rompsec セクションに対して，内蔵 ROM 周辺の配置エラー・チェックを行いません。ROM レス・モード使用時に指定することを推奨します。また，シングルチップ・モード選択時の内蔵 ROM オーバのチェックには対応していません。このオプションを指定して内蔵 ROM オーバーフローのチェックを無効とし，dump850 で確認してください。チェックしなかった場合は，rompsec セクションに対して，内蔵 ROM 周辺の配置エラー・チェックを行います。

(6) 内蔵メモリオーバーフローの警告 [-Ximem_overflow=warning]

内蔵メモリのオーバーフローのメッセージの設定を行います。チェックされている場合は，警告メッセージになります。チェックしなかった場合は，エラー・メッセージになります。

(7) コマンドラインオプション

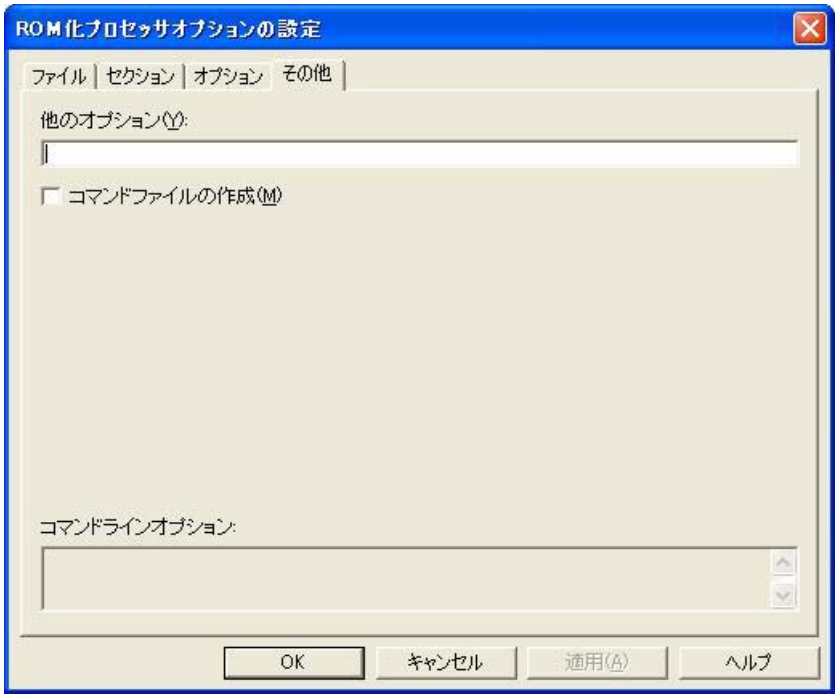
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[その他]

ROM 化プロセッサのその他の設定を行います。

図 6 － 14 [ROM 化プロセッサオプションの設定] ダイアログ：[その他] タブ



(1) 他のオプション

前記の “ ROM 化プロセッサオプションの設定 ” では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

なお，ROM 化プロセッサに関するオプションは [ROM 化プロセッサオプションの設定] ダイアログですべて指定できるため，他のオプションを使用する必要はありません。

(2) コマンドファイルの作成

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。チェックされている場合は，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，文字列の制限を意識する必要がなくなります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

なお，このオプションはデフォルトではチェックされていません。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

(3) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

第 7 章　ヘキサ・コンバータ

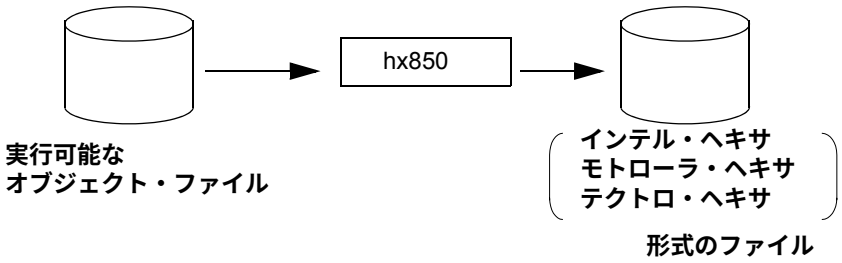
この章では，ヘキサ・コンバータ（hx850）の概要，操作方法，出力ファイルの形式について説明します。

7.1　動作の流れ

hx850 は，ROM 化プロセッサ（romp850）が出力する実行可能なオブジェクト・ファイルを入力し，ヘキサ・フォーマットに変換します。

なお，アプリケーション中に初期値ありデータがないなど，romp850 を使用する必要のないアプリケーションの場合は，リンカ（ld850）が出力する実行可能なオブジェクト・ファイルを入力します。

図 7 － 1　hx850 における動作の流れ



7.2 入出力ファイル

hx850 では，次のファイルを入力ファイルとして扱うことができます。

- *file1.out* … ld850，または romp850 によって出力された実行可能オブジェクト

ヘキサ・フォーマットの出力として，次に示す形式を指定できます。

(1) インテル・ヘキサ・フォーマット
- インテル拡張ヘキサ・フォーマット

(2) テクトロ・ヘキサ・フォーマット
- 拡張テック・ヘキサ・フォーマット

(3) モトローラ・ヘキサ・フォーマット
- S タイプ・フォーマット（スタンダード・アドレス）
- S タイプ・フォーマット（32 ビット・アドレス）

注 ヘキサ・フォーマットの各行のアドレスは，昇順で出力されます。

7.3 操作方法

この節では，hx850 の操作方法について説明します。

7.3.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
hx850 [ オプション ]... ファイル名
       [ ]    ：[ ] 内は省略できます。
       ...    ：直前の [ ] 内のパターンの繰り返しができます。
```

7.3.2 PM+ による方法

[ヘキサコンバータオプションの設定] ダイアログは，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [ヘキサコンバータオプションの設定] を選択

ヘキサ・コンバータの起動は，プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

なお，ヘキサ・コンバータが出力する出力ファイルは ld850，あるいは romp850 の出力ファイル名の拡張子を “ .hex ” に変更したファイル名となります。

また，ファイル名は，[ファイル] タブの “ 出力ファイル [-o] ” で指定することもできます。

7.4 オプションの種類と機能

次に，hx850 のオプションを示します。

〔オプション説明でのマーク〕

【PM+】	PM+ で指定項目が存在するオプション

7.4.1 ファイル

ヘキサ・コンバータが出力するファイル名を指定します。

+err_file=*file*

エラー・メッセージをファイル *file* に追加保存します。

-err_file=*file*

エラー・メッセージをファイル *file* に上書き保存します。

-o *ofile*

【PM+】

ofile という名前のファイルにヘキサ変換した結果を出力します。このオプションを省略した場合，標準出力に出力します。

7.4.2 フォーマット

ヘキサ・コンバータのフォーマット・オプションを設定します。

-b*num*

【PM+】

num で指定された値をブロック長（インテル拡張ヘキサ・フォーマット，およびモトローラ S タイプ・ヘキサ・フォーマットの場合，1 データ・レコードで示されるコードのバイト数）の最大値とします。このオプションを省略した場合，各ヘキサ・フォーマットごとに定められたデフォルトの値を用います。*num* は 10 進数，あるいは，0x，または 0X で始まる 16 進数で指定します。

表 7 － 1 ヘキサ・フォーマットのブロック／レコード

ヘキサ・フォーマット	指定できる範囲	デフォルト
インテル拡張	1 ～ 255（0x01 ～ 0xff）	31（0x1f）
モトローラ S タイプ	1 ～ 251（0x01 ～ 0xfb）	80（0x50）
モトローラ S タイプ （32 ビット・アドレス）	1 ～ 250（0x01 ～ 0xfa）	80（0x50）
拡張テック	16 ～ 255（0x10 ～ 0xff）	255（0xff）

-d*num*

【PM+】

出力するアドレスを *num* からのオフセットとします。*num* は 10 進数，あるいは，0x，または 0X で始まる 16 進数で指定します。指定可能な値は，0 ～ 0xfffffffe の範囲です。出力するアドレスは，指定した値からのオフセット値となります。デフォルトは 0 です。

-f*c*

【PM+】

文字 *c* で指定されるヘキサ・フォーマットを用います。文字 *c* の意味は次のようになっています。

I	インテル拡張
S	モトローラ S タイプ
s	モトローラ S タイプ（32 ビット・アドレス）
T	拡張テック

このオプションを省略した場合，インテル拡張ヘキサ・フォーマットになります。また，-fT オプションと -U オプションを同時に指定した場合，-U オプションの指定は無視されます。

-I*name*

【PM+】

セクション名 *name* で指定されるセクションのコードを変換し，出力します。つまり，ヘキサ・コンバータは，セグメント単位ではなくセクション単位に変換を行います。

初期値の指定されていないデータに対するセクション（セクション・タイプ NOBITS **とセクション属性** A **を持つセクション）が指定された場合，セクションのサイズ分だけ** null **文字（**\0**）を生成します。**

このオプションを省略した場合，NOBITS **以外のセクション・タイプとセクション属性** A **を持つすべてのセクションを変換します。**

hx850 **は，セグメント単位ではなくセクション単位に変換します。**

なお，セクション名に空白は使用できません。

また，このオプションと -U **オプションを同時に指定した場合，このオプションの指定は無視されます。**

-S

【PM+】

シンボル・テーブルを変換し，出力します。拡張テック・ヘキサ・フォーマットが指定された（-fT **が指定された）場合にのみ有効です。このオプションと** -U **オプションを同時に指定した場合，このオプションの指定は無視されます。**

-U

-U*num*

-U*num*,*start*,*size*

-U*start*,*size*

【PM+】

アドレス *start* からサイズ *size* で指定された領域のすべてのコードをヘキサ変換し，出力します。*start*, *size* **の指定を省略した場合，デバイス・ファイルで定義された内蔵** ROM **領域のすべてのコードをヘキサ変換し，出力します。指定された領域のうち未使用領域は *num* で満たします。*num* は，**1 **バイト，または** 2 **バイト指定ができます。*num* が** 2 **桁，または** 4 **桁に満たない場合，満たない分の** 0 **が頭に指定されたものとみなします。*num* を省略した場合，**0xff **で満たします。**

このオプションは，拡張テック・ヘキサ・フォーマット指定時に使用できません。また，このオプションを指定した場合，-I**，**-S**，**-x**，**-Z **オプションの指定は無視されます。**

-x

【PM+】

シンボル・テーブルを変換して出力する際，ローカル・シンボルも対象とします。

-S **オプションとともに指定された場合にのみ有効です。このオプションを省略した場合，グローバル・シンボルのみを対象とします。このオプションと** -U **オプションを同時に指定した場合，このオプションの指定は無視されます。**

-rom_less

【PM+】

-U オプション指定時に，デバイス・ファイルに定義された内蔵 ROM 領域の情報を使用しないようにします。また，ヘキサ変換する領域が内蔵 ROM 領域をはみ出した場合，警告メッセージの出力を行いません。このオプションと -U オプションを同時に指定する場合は，-U オプションの start, size の指定が必要になります。このオプションと -U オプションの start,size を省略した場合，デバイス・ファイルに定義された内蔵 ROM 領域が変換対象になります。また，ヘキサ変換する領域が内蔵 ROM 領域をはみだした場合は，警告メッセージを出力します。

-z

【PM+】

セクション・タイプ NOBITS とセクション属性 A を持つセクション（初期値の指定されていないデータに対するセクション，たとえば .bss セクション，および .sbss セクション）に対し，セクションのサイズ分だけ null 文字（\0）を生成します。このオプションと -U オプションを同時に指定した場合，このオプションの指定は無視されます。

7.4.3 その他

その他のオプションを設定します。

-F *devpath*

【PM+】

デバイス・ファイルをフォルダ *devpath* で探します。このオプションを省略した場合，標準フォルダで探します。

-V

hx850 のバージョン情報を標準エラー出力に出力し，終了します。

@*cfile*

cfile をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

7.5 PM+ での設定

この節では，対象プロジェクトのソース・ファイルに対して，hx850 のコマンド・オプションを設定するダイアログについて説明します。

7.5.1 [ヘキサコンバータオプションの設定] ダイアログ

[ヘキサコンバータオプションの設定] ダイアログの上部には，次の 3 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

表 7 － 2 [ヘキサコンバータ・オプションの設定] ダイアログ

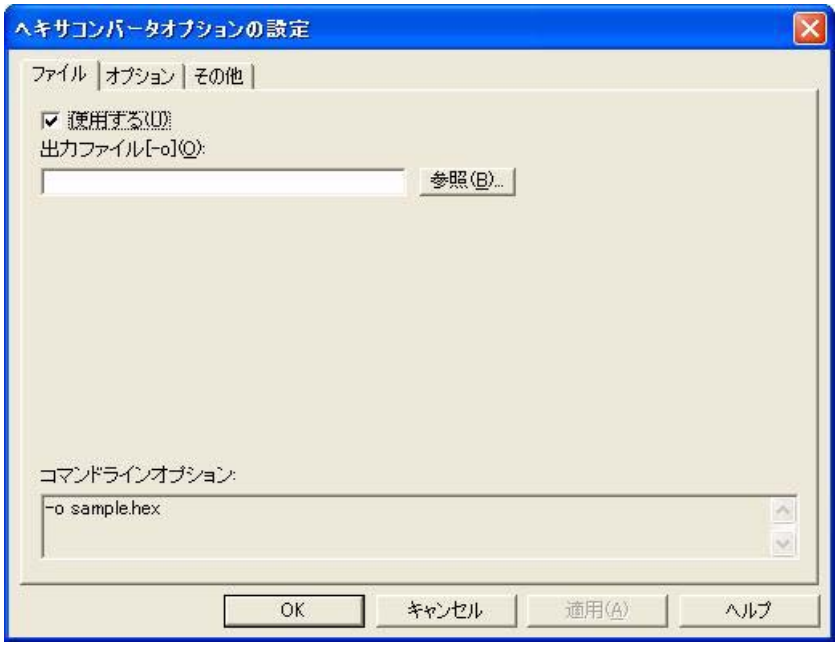
タブ	内容
[ファイル]	ファイルの設定
[オプション]	オプションの設定
[その他]	その他の設定

なお，ダイアログ上の “ [] ” 内に表示されたオプション名は，コマンド・プロンプトから起動する場合のオプション名です。

[ファイル]

ヘキサ・コンバータのファイルの設定を行います。

図 7 － 2 [ヘキサコンバータオプションの設定] ダイアログ：[ファイル] タブ



(1) 使用する

チェックされている場合は，ヘキサ・コンバータを使用します。デフォルトでチェックされています。

(2) 出力ファイル [-o]

出力するヘキサ・ファイル名を指定します。ファイル名に空白は使用できません。このオプションを省略した場合，リンカの出力ファイル名（ROM 化プロセッサまで起動する場合は ROM 化プロセッサの出力ファイル名）の拡張子を “ .hex ” に変えたファイル名が指定されたものとみなします。

また，[参照] ボタンにより表示されるダイアログで，ファイルを選択することができます。

(3) コマンドラインオプション

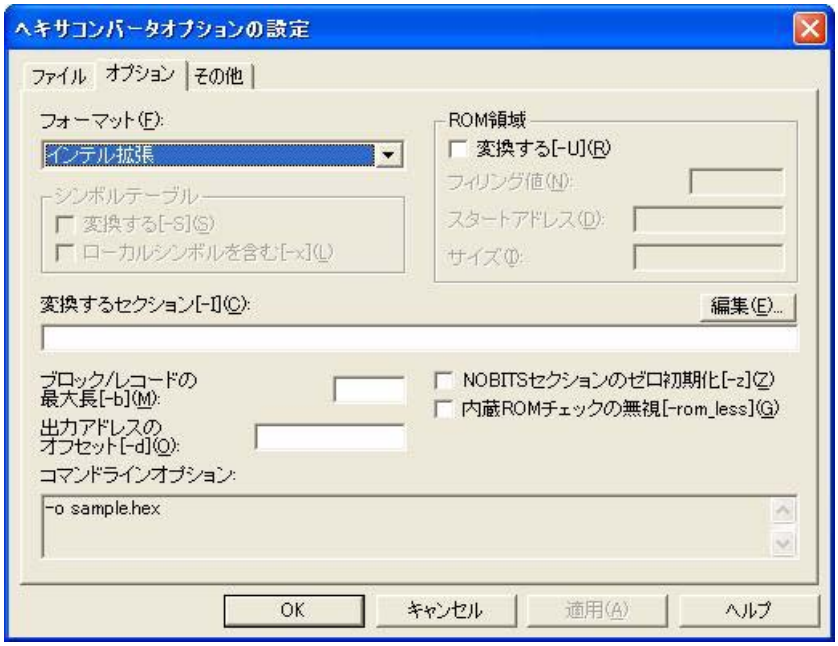
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[オプション]

ヘキサ・コンバータのオプションの設定を行います。

図 7 － 3 [ヘキサコンバータオプションの設定] ダイアログ：[オプション] タブ



(1) フォーマット

ヘキサ・ファイルの生成時に用いるヘキサ・フォーマットを指定します。指定できるフォーマットは “ インテル拡張 ”, “ モトローラタイプ S（スタンダード）”, “ モトローラタイプ S（32 ビット）”, “ 拡張テック ” です。デフォルトは , “ インテル拡張 ” です。

(2) シンボルテーブル

シンボル・テーブルを変換します。このオプションは，[フォーマット] で拡張テックを選択した場合のみ指定可能です。

(a) 変換する [-S]

シンボル・テーブルを変換し，出力します。拡張テック・ヘキサ・フォーマットが指定された（-fT が指定された）場合にのみ有効です。このオプションと -U オプションを同時に指定した場合，このオプションの指定は無視されます。

(b) ローカルシンボルを含む [-x]

シンボル・テーブル部を変換して出力する際，ローカル・シンボルも対象とします。“ 変換する [-S] ” とともに指定された場合にのみ有効です。このオプションを省略した場合，グローバル・シンボルのみを対象とします。このオプションと -U オプションを同時に指定した場合，このオプションの指定は無視されます。

(3) ROM 領域

(a) 変換する [-U]

スタートアドレスからサイズで指定された領域のすべてのコードをヘキサ変換し，出力します。スタートアドレス，サイズの指定を省略した場合，デバイス・ファイルで定義された内蔵 ROM 領域のすべてのコードをヘキサ変換し，出力します。指定された領域のうち未使用領域はフィリング値で満たします。このオプションは，拡張テック・ヘキサ・フォーマット指定時に使用できません。また，このオプションを指定した場合，

- 変換セクション指定 “ 変換するセクション [-I] ”
- シンボルの変換指定 “ 変換する [-S] ”
- シンボルテーブル内のローカルシンボル変換指定 “ ローカルシンボルを含む [-x] ”
- NOBITS 属性セクションのゼロ初期化指定 “ NOBITS セクションのゼロ初期化 [-z] ”

は無視されます。

(b) フィリング値

1 バイト，または 2 バイト指定ができます。16 進数で指定します。先頭の “ 0x ” は省略できません。2 桁，または 4 桁に満たない場合，満たない分の 0 が頭に指定されたものとみなします。省略した場合，0xff で満たします。

(c) スタートアドレス

変換する領域のスタートアドレスを 10 進数，または 16 進数で指定します。“ サイズ ” を指定した場合は，必ず指定してください。

(d) サイズ

変換する領域のサイズを 10 進数，または 16 進数で指定します。“ スタートアドレス ” を指定した場合は，必ず指定してください。

(4) 変換するセクション [-I]

ヘキサ・フォーマットに変換するセクションを指定します。ヘキサ・コンバータは，セグメント単位ではなくセクション単位に変換を行います。初期値の指定されていないデータに対するセクション（セクション・タイプ NOBITS とセクション属性 A を持つセクション）が指定された場合，セクションのサイズ分だけ null 文字（\0）を生成します。このオプションを省略した場合，NOBITS 以外のセクション・タイプとセクション属性 A を持つすべてのセクションを変換します。複数のセクションを指定する場合，“ ; （セミコロン）” で区切ります。

[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，セクション項目をダイアログ上で編集することができます。

なお，セクション名に空白は使用できません。

(5) ブロック／レコードの最大長 [-b]

指定された値をブロック長（インテル拡張ヘキサ・フォーマット，およびモトローラ S タイプ・ヘキサ・フォーマットの場合，1 データ・レコードで示されるコードのバイト数）の最大値とします。このオプションを省略した場合，各ヘキサ・フォーマットごとに定められたデフォルトの値を用います。10 進数，あるいは，0x，または 0X で始まる 16 進数で指定します。

表 7 － 3 ヘキサ・フォーマットのブロック／レコード

ヘキサ・フォーマット	指定できる範囲	デフォルト
インテル拡張	1 ～ 255（0x01 ～ 0xff）	31（0x1f）
モトローラ S タイプ	1 ～ 251（0x01 ～ 0xfb）	80（0x50）
モトローラ S タイプ （32 ビット・アドレス）	1 ～ 250（0x01 ～ 0xfa）	80（0x50）
拡張テック	16 ～ 255（0x10 ～ 0xff）	255（0xff）

(6) 出力アドレスのオフセット [-d]

出力するアドレスのオフセットを，10 進数，または 0x で始まる 16 進数で指定します。出力するアドレスは，指定した値からのオフセット値となります。デフォルトは 0 です。

(7) NOBITS セクションのゼロ初期化 [-z]

セクション・タイプ NOBITS とセクション属性 A を持つセクション（初期値の指定されていないデータに対するセクション，たとえば .bss セクション，および .sbss セクション）に対し，セクションのサイズ分だけ null 文字（\0）を生成します。このオプションと -U オプションを同時に指定した場合，このオプションの指定は無視されます。

(8) 内蔵 ROM チェックの無視 [-rom_less]

-U オプション指定時に，デバイス・ファイルに定義された内蔵 ROM 領域の情報を使用しないようにします。また，ヘキサ変換する領域が内蔵 ROM 領域をはみ出した場合，警告メッセージの出力を行いません。このオプションと -U オプションを同時に指定する場合は，-U オプションの start, size の指定が必要になります。このオプションと -U オプションの start,size を省略した場合，デバイス・ファイルに定義された内蔵 ROM 領域が変換対象になります。また，ヘキサ変換する領域が内蔵ROM領域をはみだした場合は，警告メッセージを出力します。

(9) コマンドラインオプション

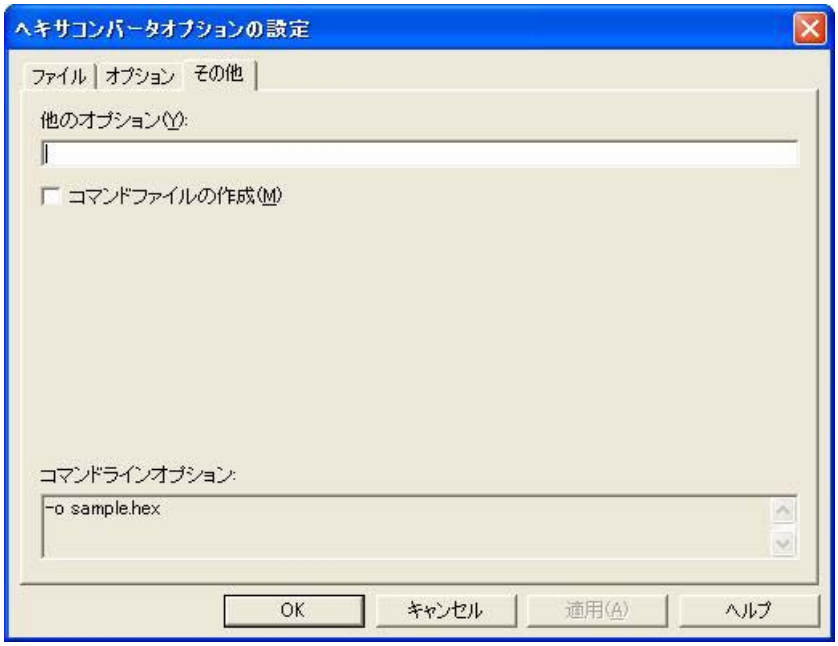
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[その他]

ヘキサ・コンバータのその他の設定を行います。

図 7 － 4 [ヘキサコンバータオプションの設定] ダイアログ：[その他] タブ



(1) 他のオプション

前記の “ ヘキサコンバータの設定 ” では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

なお，ヘキサ・コンバータに関するオプションは “ ヘキサコンバータの設定 ” ですべて指定できるため，他のオプションを使用する必要はありません。

(2) コマンドファイルの作成

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。チェックされている場合は，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，文字列の制限を意識する必要がなくなります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

なお，このオプションはデフォルトではチェックされていません。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

(3) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

7.6 出力形式

この章では，hx850 の出力ファイルの形式について説明します。

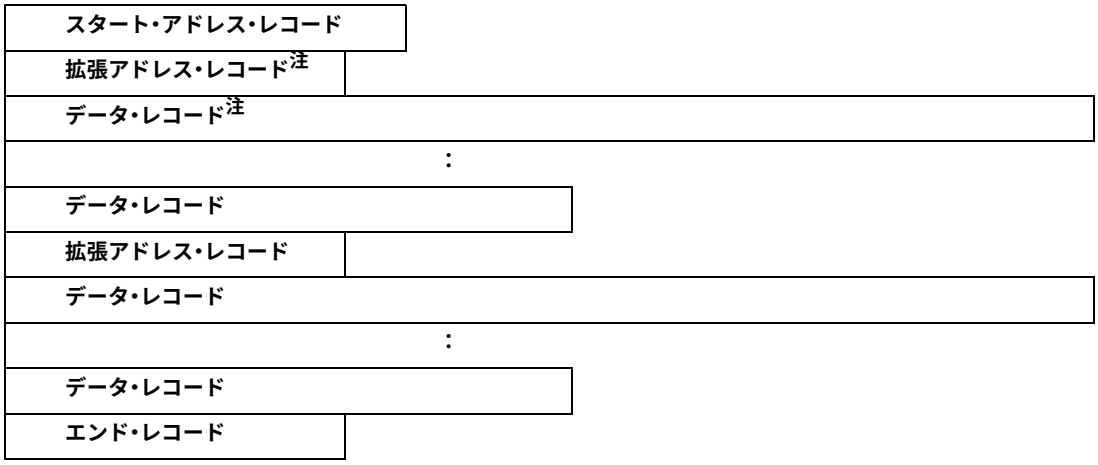
7.6.1 インテル拡張

インテル拡張ヘキサ・フォーマットのファイルは，スタート・アドレス・レコード，拡張アドレス・レコード，データ・レコード，およびエンド・レコードの 4 種類のレコード[注]により構成されます。

注 各レコードは，ASCII コードで出力されます。

次図にインテル拡張ヘキサ・フォーマットのファイル構成を示します。

図 7 － 5 インテル拡張ヘキサ・フォーマットのファイル構成



注 拡張アドレス・レコード，およびデータ・レコードは繰り返されます。

各レコードは，各種フィールドにより次の形に構成されます。

:	CC	AAAA	TT	[フィールド]...	SS	NL
(a)	(b)	(c)	(d)		(e)	(f)

(a) レコード・マーク

(b) バイト数（[フィールド]... の 2 桁ずつの 16 進数で表されるバイトのバイト数）

(c) ロケーション・アドレス

(d) レコード・タイプ（03：スタート・アドレス・レコード，02：拡張アドレス・レコード，00：データ・レコード，01：エンド・レコード）

(e) チェック・サム（：，SS,，NL を除くレコード内の 2 桁ずつの 16 進数で表される値を初期値 0 から順に減算し，その下位 1 バイトを 2 桁の 16 進数で表したもの）

(f) ニュー・ライン（\n）

(1) スタート・アドレス・レコード

エントリ・ポイント・アドレスを示します。

```
:    04    0000    03    PPPP    OOOO    SS    NL
     (a)   (b)     (c)   (d)     (e)
```

(a) バイト数は 04 で固定

(b) 0000 で固定

(c) レコード・タイプは 03

(d) エントリ・ポイント・アドレスのパラグラフ値[注]

(e) エントリ・ポイント・アドレスのオフセット値

注 アドレスは（パラグラフ値 <<4）+ オフセット値で求められます。

(2) 拡張アドレス・レコード

ロード・アドレスのパラグラフ値を示します[注]。

注 （データ・レコードを出力する際）セグメントの先頭で，またはデータ・レコードのロード・アドレスのオフセット値が最大値 0xffff を越えてセグメントが新しくなる際，出力されます。

```
:    02    0000    02    PPPP    SS    NL
     (a)   (b)     (c)   (d)
```

(a) バイト数は 02 で固定

(b) 0000 で固定

(c) レコード・タイプは 02

(d) セグメントのパラグラフ値

(3) データ・レコード

コードの値を示します。

:	CC	AAAA	00	DD...DD	SS	NL
	(a)	(b)	(c)	(d)		

(a) バイト数[注]

(b) ロケーション・アドレス

(c) レコード・タイプは 00

(d) コード（コードの 1 バイトごとを 2 桁の 16 進数で表したもの）

注 0x1 から 0xff までの範囲に限られます（1 つのデータ・レコードで示されるコードのバイト数の最小値は 1 で最大値は 255 です）。

例

:	04	0100	00	3C58E01B	6C	NL
	(a)	(b)	(c)	(d)	(e)	

(a) 3C58E01B の 2 桁ずつの 16 進数で表されるバイトのバイト数は 04

(b) ロケーション・アドレスは 0100

(c) レコード・タイプは 00

(d) コード

(e) チェック・サムは 04 + 01 + 00 + 00 + 3C + 58 + E0 + 1B = 194 の 2 の補数 E6 の下位 1 バイトを 2 文字の 16 進数で表したもので 6C

(4) エンド・レコード

コードの終わりを示します。

:	00	0000	01	FF	NL
	(a)	(b)	(c)	(d)	

(a) バイト数は 00 で固定

(b) 0000 で固定

(c) レコード・タイプは 01

(d) チェック・サムは FF で固定

[参考] インテルヘキサについて

インテルヘキサ・フォーマットのロケーション・アドレスは 2 **バイト（**16 **ビット）です。したがって，**64K **の空間しか直接指定はできません。それを拡張するために，**16 **ビットの拡張アドレスを追加して** 1M **（**20 **ビット）の空間まで扱えるようにしたのがインテル拡張ヘキサ・フォーマットです。**

具体的には，16 **ビットの拡張アドレスを指定するレコードタイプを追加しています。この追加された拡張アドレスの** 4 **ビットをシフトしてロケーション・アドレスと加算することで，**20 **ビットのアドレスを表現できるようになっています。**

たとえば，FFFFFH **を示す場合には，拡張アドレスに** F000H **を設定し，ロケーション・アドレスに** FFFFH **を指定します。**

このようにインテル拡張ヘキサ・フォーマットでは 0 **～** FFFFFH **までしかアドレッシングできません。**FFFFFH **のような場合には別のオブジェクト形式を使用する必要があります。**

hx850 **では，このアドレス，サイズにおいて，このフォーマットの規定に違反していた場合，メッセージを出力します。**

インテル拡張ヘキサの場合，表現可能な値が 20 **ビット，つまり，**1M（0x100000）**バイトになります。**

```
W8737 : The start address of convert area exceeds the maximum value of the
address that can be expressed in the Intel expanded hex format
```

“ W8737 **”のメッセージが出た場合は，ヘキサ変換する領域のスタート・アドレスが，**1 M **バイトを越えている場合になります。**

```
W8735: The address of convert area exceeds the maximum value of the address
that can be expressed in the Intel expanded hex format
```

“ W8735 **”のメッセージが出た場合は，ヘキサ変換を行おうとするアドレスが** 1M **（**20 **ビット）を越えた場合になります。**

次の例のような場合には，1M **を越えなくても，上記エラーが発生します。**

例

- -d **オプションで指定したアドレスを起点とするオフセットとしない**
  → 絶対アドレスをヘキサ・フォーマットに格納する
- 20 **ビットで表現可能なアドレスの上限付近にセクションを配置**
  →スタート・アドレスは 20 **ビットに収まっているが，セクションの途中から** 20 **ビットを越える**

この 2 **バターンに合致する場合には，変換する領域がわずかに** 4 **バイトであるとしても，“** W8735 **”の メッセージ が発生します。**

7.6.2 モトローラ S タイプ

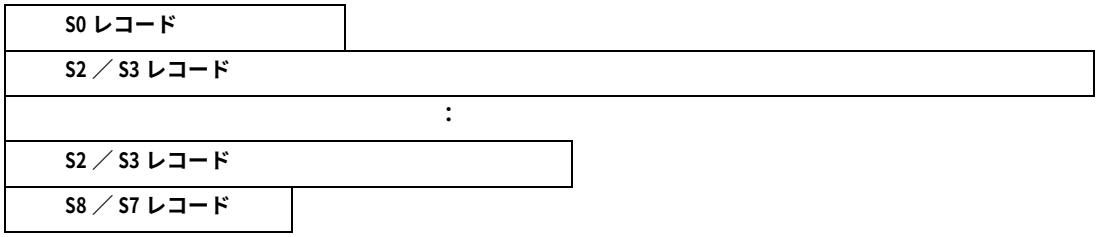
モトローラ S タイプ・ヘキサ・フォーマットのファイルは，ヘッダ・レコードである S0 レコード，データ・レコードである S2 ／ S3 レコード，エンド・レコードである S8 ／ S7 レコードの 5 種類のレコード[注 1] により構成されます[注 2]。

次図にモトローラ S タイプ・ヘキサ・フォーマットのファイル構成を示します。

注 1 各レコードは，ASCII コードで出力されます。

注 2 モトローラ S タイプ・ヘキサ・フォーマットには，（24 ビット）スタンダード・アドレスのものと，32 ビット・アドレスのものとが存在し，スタンダード・アドレスのフォーマットは S0，S2，および S8 レコード，32 ビット・アドレスのフォーマットは S0，S3，および S7 レコードによって構成されます。

図 7 － 6 モトローラ S タイプ・ヘキサ・フォーマットのファイル構成



各レコードは，各種フィールドにより次の形に構成されます。

ST	LL	フィールド [フィールド]...	SS	NL
(a)	(b)		(c)	(d)

(a) レコード・タイプ

(b) レコード長（フィールド [フィールド]... の 2 桁ずつの 16 進数で表されるバイトのバイト数 + SS で表されるバイト数[注]）

(c) チェック・サム（ST，SS，NL を除くレコード内の 2 桁ずつの 16 進数で表されるバイトの値を合計したものの 1 の補数を取り，その下位 1 バイトを 2 桁の 16 進数で表したもの）

(d) ニュー・ライン（\n）

注 1 です。

(1) S0 レコード

ファイル名を示します。

S0	LL	FF...FF	SS	NL
(a)		(b)		

(a) レコード・タイプは S0

(b) ファイル名（指定されたファイル名の ASCII コード表示）

(2) S2 レコード

コードの値を示します。

S2	LL	AAAAAA	DD...DD	SS	NL
(a)		(b)	(c)		

(a) レコード・タイプは S2

(b) ロード・アドレス（24 ビット[注]）

(c) コード（1 バイトごとを 2 桁の 16 進数で表したもの）

注 0x0 ～ 0xffffff の範囲です。

(3) S3 レコード

コードの値を示します。

S3	LL	AAAAAAAA	DD...DD	SS	NL
(a)		(b)	(c)		

(a) レコード・タイプは S3

(b) ロード・アドレス（32 ビット[注]）

(c) コード（1 バイトごとを 2 桁の 16 進数で表したもの）

注 0x0 ～ 0xffffffff の範囲です。

(4) S7 レコード

エントリ・ポイント・アドレスを示します。

S7	LL	AAAAAAAA	SS	NL
(a)		(b)		

(a) レコード・タイプは S7

(b) エントリ・ポイント・アドレス（32 ビット[注]）

注 0x0 ～ 0xffffffff の範囲です。

(5) S8 レコード

エントリ・ポイント・アドレスを示します。

S8	LL	AAAAAA	SS	NL
(a)		(b)		

(a) レコード・タイプは S8

(b) エントリ・ポイント・アドレス（24 ビット[注]）

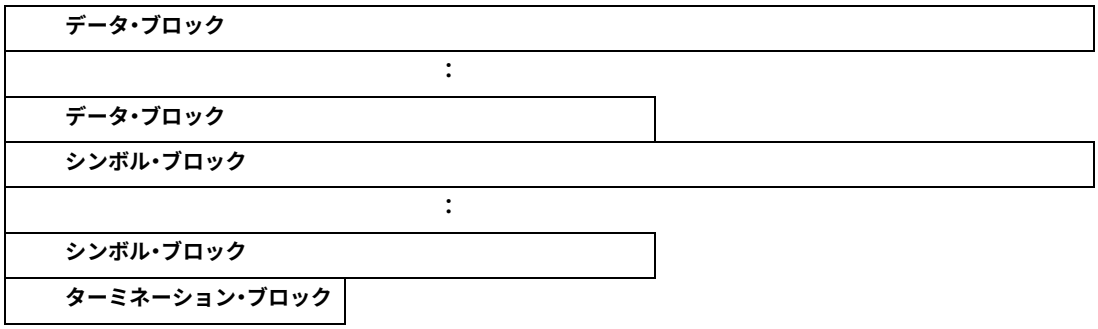
注 0x0 ～ 0xffffff の範囲です。

7.6.3 拡張テック

拡張テック・ヘキサ・フォーマットのファイルは，データ・ブロック，シンボル・ブロック，およびターミネーション・ブロックの 3 種類のブロックにより構成されます。

次図に拡張テック・ヘキサ・フォーマットのファイル構成を示します。

図 7 － 7 拡張テック・ヘキサ・フォーマットのファイル構成



各ブロックは，各種フィールドにより次の形に構成されます。

%	LL	T	SS	フィールド [フィールド]...	NL
(a)	(b)	(c)	(d)		(e)

(a) ヘッダ文字

(b) ブロック長（%,NL を除くブロック内のすべての文字の数）

(c) ブロックの種類[注 1]

(d) チェック・サム（%，SS，NL を除くブロック内のすべての文字に対する値[注 2] を合計したものを 256 で割った余りの値を，2 桁の 16 進数で表したもの）

(e) ニュー・ライン（\n）

注 1 6 … データ・ブロック，3 … シンボル・ブロック，8 … ターミネーション・ブロックです。

注 2 各文字に対する値は次のように定められています（0 ～ 9：0 ～ 9，A ～ Z：10 ～ 35，$：36，%：37，.：38，-：39，a ～ z：40 ～ 65）。

(1) **データ・ブロック**

コードの値を示します。

%	LL	T	SS	LA...A	D...D	NL
		(a)		(b)	(c)	

(a) **ブロックの種類は** 6

(b) **ロード・アドレスの桁数とロード・アドレス**

(c) **コード（コードの** 1 **バイトごとを** 2 **桁の** 16 **進数で表したもの）**

例

%	15	6	1C	3 100	020202020202	NL
	(a)	(b)	(c)	(d)	(e)	

(a) **ブロック長は** 15

(b) **ブロックの種類は** 6

(c) **チェック・サムは**

1 + 5 + 6 + 3 + 1 + 0 + 0 + 0 + 2 + 0 + 2 + 0 + 2 + 0 + 2 + 0 + 2 + 0 + 2 = 28 **を** 256 **で割った余りの値を** 2 **桁の** 16 **進数で表したもので** 1C

(d) **ロード・アドレスの桁数は** 3**，ロード・アドレスは** 100

(e) **コード**

(2) **シンボル・ブロック**

シンボルの値を示します。

```
%     LL     T     SS     L N...N  [SEDF 注]      SYDF [SYDF]    NL
             (a)          (b)      (A)            (B)
```

(a) **ブロックの種類は** 3

(b) **セクション名の文字数とセクション名**

注 **セクションごとに** 1 **つ存在しなければならず，任意の数のシンボル定義フィールドの前，または後ろに続けることができます。**

(A) **セクション定義フィールド**（SEDF）

```
0      L B...B  L L...L
(a)    (b)      (c)
```

(a) **このフィールドがセクション定義フィールドであることを示す**

(b) **セクションのベース・アドレスの桁数とセクションのベース・アドレス**

(c) **セクションの長さの桁数とセクションの長さ**

(B) **シンボル定義フィールド**（SYDF）

```
T      L S...S  L V...V
(a)    (b)      (c)
```

(a) **シンボルの種類**

1：**グローバル・アドレス（バインディング・クラス** GLOBAL **と** ABS **以外のタイプを持つシンボル）**

2：**グローバル・スカラ（バインディング・クラス** GLOBAL **とタイプ** ABS **を持つシンボル）**

5：**ローカル・アドレス（バインディング・クラス** LOCAL **と** ABS **以外のタイプを持つシンボル）**

6：**ローカル・スカラ（バインディング・クラス** LOCAL **とタイプ** ABS **を持つシンボル）**

(b) **シンボルの文字数とシンボル**

(c) **シンボルの値の桁数とシンボルの値**

例 1

```
%  37  3   60  8SVCSTUFF0  2402C6  22CR1D14OPEN25014READ25815WRITE260  NL
   (a) (b) (c) (d)          (e)     (f)
```

(a) **ブロック長は** 37

(b) **ブロックの種類は** 3

(c) **チェック・サムは** 60

(d) **セクション名の文字数は** 8，**セクション名は** SVCSTUFF

(e) **セクション定義フィールド（セクションのベース・アドレスの桁数は** 2，**セクションのベース・アドレスは** 20，**セクションの長さの桁数は** 2，**セクションの長さは** C6）

(f) **シンボル定義フィールド**（22CR1D ／ 14OPEN250 ／ 14READ258 ／ 15WRITE260）

例 2

```
%  37  3   C8  8SVCSTUFF0  15CLOSE26814EXIT27029BUFLENGTH28013BUF278   NL
   (a) (b) (c)  (d)                (e)
```

(a) **ブロック長**

(b) **ブロックの種類は** 3

(c) **チェック・サム**

(d) **セクション名の文字数は** 8**，セクション名は** SVCSTUFF

(e) **シンボル定義フィールド**

（15CLOSE268 ／ 14EXIT270 ／ 29BUFLENGTH280 ／ 13BUF278**）**

(3) **ターミネーション・ブロック**

エントリ・ポイント・アドレスを示します。

%	LL	T	SS	L A...A	NL
			(a)	(b)	

(a) **ブロックの種類は** 8

(b) **エントリ・ポイント・アドレスの桁数とエントリ・ポイント・アドレス**

例

%	08	8	1A	2 80	NL
	(a)	(b)	(c)	(d)	

(a) **ブロック長は** 8

(b) **ブロックの種類は** 8

(c) **チェック・サムは** 1A

(d) **エントリ・ポイント・アドレスの桁数は** 2，**エントリ・ポイント・アドレスは** 80

第 8 章　アーカイバ

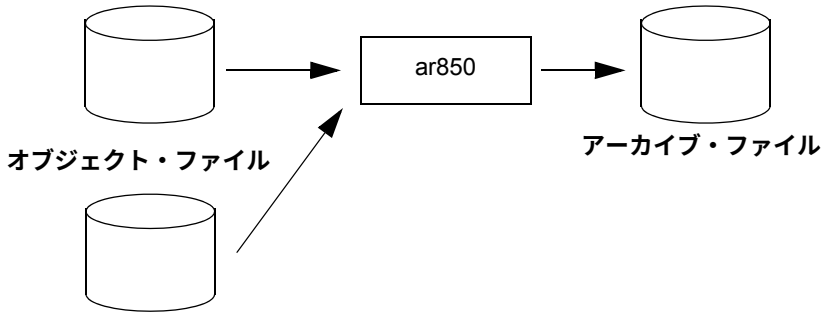
この章では，アーカイバ（ar850）の概要，操作方法について説明します。

8.1　アーカイバとは

アーカイバとは，指定されたリロケータブルなオブジェクト・ファイルを結合し，1 つのアーカイブ・ファイルを生成するユーティリティです。つまり，ユーザが複数のオブジェクトをまとめ，“ ライブラリ ” として作成する場合に使用します。

C コンパイラ・パッケージに入っている “ ar850 ” がアーカイバです。

図 8 － 1　ar850 における動作の流れ



ar850 によって生成されるアーカイブ・ファイルは，リンク時の入力ファイルとして指定できます。アーカイブ・ファイルが指定された場合，ld850 は指定されたアーカイブ・ファイル内から必要とされるオブジェクトを検索し，見つかったオブジェクトのみをリンクします。

8.2 操作方法

この節では，ar850 の操作方法について説明します。

8.2.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
ar850 [エラー出力指定オプション] キー [オプション] [メンバ名注] アーカイブ・ファイル名
[メンバ名，またはファイル名]...
        [ ]    ：[ ]内は省略できます。
        ...    ：直前の[ ]内のパターンの繰り返しができます。
```

注　ファイルは，アーカイブ・ファイル内に連結されるとメンバと呼ばれます。メンバは，連結される前のファイルのときと同じ名前を持ちます。

8.2.2 PM+ による方法

PM+ でアーカイブ・ファイル（ライブラリ・ファイル）を生成するときは，専用のプロジェクトを作成します。

PM+ でプロジェクトを設定する際は，“ マイクロコントローラ名 ” として次のいずれかを選択します。

- V850 Common Library
- V850 Library
- V850E Core Common Library
- V850E2 Core Common Library

これによって，アーカイブ・ファイル（ライブラリ・ファイル）を生成するプロジェクトが作成されます。

また，[ツール] メニューで選択できる設定ダイアログは次のとおりとなります。

- “ [コンパイラ共通オプションの設定] ダイアログ ”
- “ [コンパイラオプションの設定] ダイアログ ”
- “ [アセンブラオプションの設定] ダイアログ ”
- “ [アーカイバオプションの設定] ダイアログ ”
- “ [セクションファイルジェネレータオプションの設定] ダイアログ ”
- “ [静的性能解析ツール] ダイアログ ”

なお，アーカイバの起動は，プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

コマンド・ライン上から ar850 を起動する場合，オブジェクト・ファイル群をまとめて，アーカイブ・ファイルを作成します。また，アーカイブ・ファイル内のオブジェクトの操作など，細かい作業を行うことができます。

一方，PM+ を使用してアーカイブ・ファイルを生成する場合，ソース・ファイルをコンパイル，アセンブルし，作られたオブジェクトをアーカイブ・ファイルにまとめるという作業になります。PM+ 上からは完成されたアーカイブ・ファイルに対して操作することはできません。そのためコマンド・ラインと PM+ を使い分けていく必要があります。

これらのことから，コマンド・ライン上でのオプション設定と，PM+ 上での設定項目には違いがあります。それぞれについて説明していきます。

8.3 キー／オプションの種類と機能

“ キー ” と “ オプション ” の種類と機能について説明します。キーとは，起動時に必ずどれかが指定されていなければならないもので，オプションとは省略することができるものです。

〔キー／オプション説明でのマーク〕

【PM+】	PM+ で指定項目が存在するキー／オプション

8.3.1 キーの種類と機能

次に，ar850 のキー一覧を示します。

v

ar850 のバージョン情報を標準出力に出力し，終了します。

d

指定されたメンバを指定されたアーカイブ・ファイルから削除します。

m

指定されたメンバを指定されたアーカイブ・ファイルの最後に移動します。

ma *member*

指定されたメンバを指定されたアーカイブ・ファイル内のメンバ *member* の直後に移動します。

member を省略した場合，処理を中止します。

mb *member*

指定されたメンバを指定されたアーカイブ・ファイル内のメンバ *member* の直前に移動します。

member を省略した場合，処理を中止します。

q

指定されたファイルを指定されたアーカイブ・ファイルの最後に追加します。指定されたファイルと同名のメンバが存在しているかどうかのチェックは行いません。指定されたアーカイブ・ファイルが存在しない場合，指定されたファイルを含んだアーカイブ・ファイルを新規に生成します。指定されたファイルと同名のメンバのチェックは行いません。同名のメンバが存在した場合，アーカイブ・ファイルに複数の同名メンバが含まれ，リンク時には最も古いメンバが選択されます。

新規作成を行う場合は，古いアーカイブ・ファイルを必ず削除してください。同名のメンバを入れ替える場合には，r キーを使用してください。

r

指定されたファイルを指定されたアーカイブ・ファイル内の同名のメンバと入れ替えます。指定されたアーカイブ・ファイル内に同名のメンバが存在しない場合，指定されたファイルを指定されたアーカイブ・ファイルの最後に追加します。指定されたアーカイブ・ファイルが存在しない場合，指定されたファイルを含んだアーカイブ・ファイルを新規に生成します。

ra *member*

指定されたファイルを指定されたアーカイブ・ファイル内の同名のメンバと入れ替え，メンバ *member* の直後に移動します。指定されたアーカイブ・ファイル内に同名のメンバが存在しない場合，指定されたファイルを指定されたアーカイブ・ファイルの最後に追加します。*member* を省略した場合，処理を中止します。

ru

【PM+】

指定されたファイルが指定されたアーカイブ・ファイル内の同名のメンバより最近に更新されている場合，入れ替えを行います。指定されたアーカイブ・ファイル内に同名のメンバが存在しない場合，指定されたファイルを指定されたアーカイブ・ファイルの最後に追加します。指定されたアーカイブ・ファイルが存在しない場合，指定されたファイルを含んだアーカイブ・ファイルを新規に生成します。

t

メンバ名が指定された場合，指定されたアーカイブ・ファイル内に存在しているメンバのメンバ名のみを出力します。メンバ名が指定されなかった場合，指定されたアーカイブ・ファイル内に存在しているすべてのメンバのメンバ名を標準出力に出力します。

x

メンバ名が指定された場合，指定されたメンバが指定されたアーカイブ・ファイル内に存在していればそのメンバを取り出し，同名のファイルを生成します。メンバ名が指定されなかった場合，指定されたアーカイブ・ファイル内に存在しているすべてのメンバを取り出し，同名のファイルを生成します。アーカイブ・ファイルの内容は変更されません。

8.3.2 オプションの種類と機能

オプションは省略することができます。

(1) オプション一覧

c

【PM+】

メッセージを出力しません。

v

【PM+】

アーカイバの実行状況を “ [a|d|q|m|r|x] - *file* ” の形式で出力します。

a - *file*	追加
d - *file*	削除
q - *file*	新規作成
m - *file*	移動
r - *file*	置換
x - *file*	抽出

@cfile

【PM+】

cfile をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

(2) エラー出力指定オプション

+err_file=*file*

エラー・メッセージをファイル *file* に追加保存します。

-err_file=*file*

エラー・メッセージをファイル *file* に上書き保存します。

8.4 PM+ での設定

この節では，対象プロジェクトのソース・ファイルに対して，ar850 のコマンド・オプションを設定するダイアログについて説明します。

8.4.1 [アーカイバオプションの設定] ダイアログ

[アーカイバオプションの設定] ダイアログの上部には，次の 1 つのタブが表示されています。

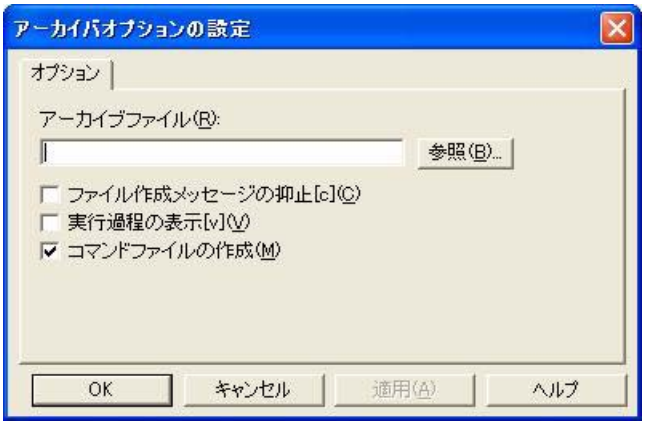
表 8 － 1 [アーカイバオプションの設定] ダイアログ

タブ	内容
[オプション]	オプションの設定

[オプション]

アーカイバのオプションの設定を行います。

図 8 － 2 [アーカイバオプションの設定] ダイアログ：[オプション] タブ



(1) アーカイブファイル

出力するアーカイブ・ファイル名（拡張子は“ .a ”）を指定します。このオプションを省略した場合，プロジェクト・ファイル名の拡張子を “ .a ” に変えたファイル名が指定されたものとみなします。また，[参照] ボタンにより表示されるダイアログで，ファイルを選択することができます。

(2) ファイル作成メッセージの抑止

ファイル作成時のメッセージを出力しません。

(3) 実行過程の表示

実行状況を次の形式で出力します。

“ a - ファイル名 ”… 追加
“ d - ファイル名 ”… 削除
“ q - ファイル名 ”… 新規作成
“ m - ファイル名 ”… 移動
“ r - ファイル名 ”… 置換
“ x - ファイル名 ”… 抽出

(4) 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・ボックスはデフォルトで選択されています。

第 9 章　セクション・ファイル・ジェネレータ

この章では，セクション・ファイル，セクション・ファイル・ジェネレータ（sf850）の概要，セクション・ファイルの利用手順，sf850 の操作方法について説明します。

9.1　セクション・ファイル

この節では，セクション・ファイルについて説明します。

セクション・ファイルとは，C 言語ソース・ファイル内で宣言されている外部変数（グローバル変数），静的変数（スタティック変数）を配置するセクションを定義するファイルです。コンパイル時にセクション・ファイルを参照することにより，これらの変数を配置するセクションを決定できます。デフォルトでは，V850 マイクロコントローラの内蔵 RAM 領域に配置するセクションである .tidata 属性，.tidata.word 属性，.tidata.byte 属性のセクションに，アクセス頻度の高い変数をできるだけ多く割り当てるようにしています。

ca850 では，C 言語ソース・ファイルにて外部変数を宣言し，それを意図したセクションに配置する方法には，次の 3 種類があります。

(1)　コンパイラ・オプション（-G*num*）により，.sdata セクション／ .sbss セクションへデータ・サイズを限定して配置する
(2)　#pragma section 指令により，変数ごとに配置するセクションを決定する
(3)　セクション・ファイルにより，コンパイラ起動時に指定した変数を配置する

(1) の方法は，あるサイズ以下の外部変数を .sdata か .sbss のどちらかに配置できればよい場合に適した方法です。コンパイル・オプションにて指定するため，C 言語ソース・ファイルに変更を加える必要はありません。

配置セクションをもっと自由に設定したい場合は，(2) の方法のように C 言語ソース・ファイルにて“ #pragma section ”指令を使用し，明示的に配置セクションを指定します。しかし，この場合は C 言語ソース・ファイルに変更を加える必要があります。

配置セクションを自由に設定したいが，たとえば，ANSI に厳密に沿った記述をするために，#pragma section 指令を使用したくないときや，以前 ca850 以外でコンパイルした C 言語ソース・ファイルを，ソースにあまり変更を加えずに ca850 用に移植したいときなどは，(1) や (2) の方法はあまり使用できません。

これを解決するのが，(3) の「セクション・ファイル」を使用する方法です。

セクション・ファイルにて，

- 静的変数の場合，その変数が宣言されている C 言語ソース・ファイル名
- 外部変数名，静的変数名とそれらを配置したいセクション名

をすべての外部変数，静的変数に対して定義します。そして，ca850 にセクション・ファイルを参照させることにより，C 言語ソース・ファイルに修正を加えることなく，これらの変数を意図した場所に配置できます。

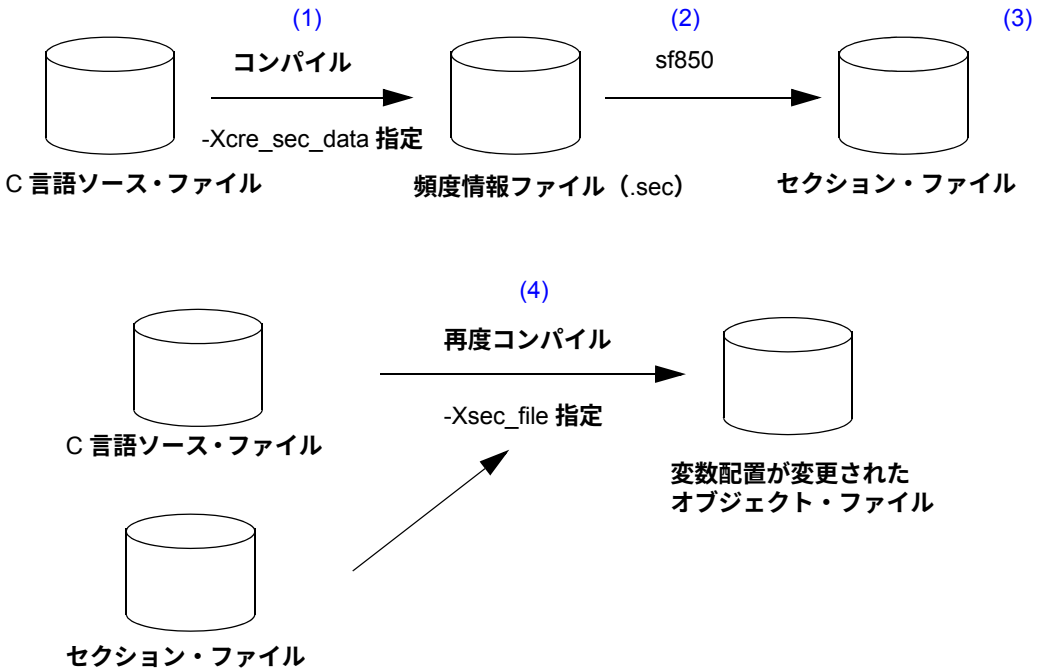
ca850 では，コンパイルのオプション指定（-Xcre_sec_data，または -Xcre_sec_data_only オプション）により，頻度情報ファイルが生成されますが，このファイルを「セクション・ファイル・ジェネレータ（sf850）」に入力することによってセクション・ファイルを生成します。

ただし，sf850 は V850 マイクロコントローラの内蔵 RAM への配置を目的とした “ tidata 属性 ”“ tidata.word 属性 ”“ tidata.byte 属性 ” のセクションにデータを配置するための情報を出力する仕様になっています。

なお，セクション・ファイルはテキスト形式のファイルなので，エディタなどで編集して変更することができます。つまり，sf850 によって出力されたセクション・ファイルに変更を加え，最終的なセクション・ファイルを作成していきます。

完成したセクション・ファイルを使用（-Xsec_file オプション指定）して，もう一度コンパイルを行うと，外部変数，静的変数が指定したセクションに配置されたオブジェクトが完成します。

図 9 － 1 セクション・ファイル指定によるコンパイルのイメージ



(1) セクション・ファイルを作成するために，-Xcre_sec_data オプション指定で一度コンパイルします。

(2) sf850 で頻度情報ファイルをセクション・ファイルに変換します。

(3) 必要ならばセクション・ファイルを編集します。

(4) セクション・ファイルを入力するため -Xsec_file オプション指定で再度コンパイルします。

セクション・ファイルの書式については，「9.2 セクション・ファイルの書式」を参照してください。

セクション・ファイルにて配置指定ができる変数は，外部変数（グローバル変数），ファイル内静的変数（ファイル内で宣言されたスタティック変数），関数内静的変数（関数内で宣言されたスタティック変数）です。文字列定数（“ abc ” など）を配置指定することはできません。

複数の C 言語ソース・ファイルを個々にコンパイルし，リンクしてオブジェクト・ファイルを生成する場合，それぞれに頻度情報出力を指定してコンパイルし，複数の .sec ファイルを生成することになります。ただし，セクション・ファイル生成時は，すべての .sec ファイルを一度に sf850 に入力し，統合する必要があります。そうしないと，外部変数に対して変数情報が統合されず，有効なセクション・ファイルを生成できません。

セクション・ファイルで指定した変数は“ #pragma section ”指令でセクション配置指定したことと同じになります。したがって，外部変数の仮定義は“ 定義 ”として扱われるため，複数ファイルで仮定義した場合，リンク時にエラーとなります。そのような場合，外部変数を参照するファイルでは，必ず extern 宣言する必要があります。

なお，セクション・ファイルで配置指定した変数が C 言語ソース・ファイル内で #pragma section 指令によって違うセクションへの配置指定がされていた場合，セクション・ファイルによる指定が優先されます。

また，コンパイラ・オプションで“ -G*num* ”が指定されていても，セクション・ファイルでその変数が .sdata ／ .sbss セクションへの配置を指定されていれば，*num* の大きさに関係なく，.sdata ／ .sbss セクションへ配置されます。つまり，「セクション・ファイル指定」「#pragma section 指定」「-G*num* 指定」の優先度は次のようになります。

（優先度高）セクション・ファイル指定 ＞ `#pragma section` 指定 ＞ `-G`*num* 指定（優先度低）

9.2 セクション・ファイルの書式

セクション・ファイルは，コンパイル時に入力し，変数の配置セクションを変更するテキスト・ファイルです。これにより，C 言語ソース・ファイルに変更を加えずに，変数の配置を変更できます。セクション・ファイルによる配置指定は，C 言語ソース・プログラムにおける #pragma section 指令よりも優先されます。

なお，ca850 では，コンパイル時に sf850 により出力されたセクション・ファイルを指定できます。sf850 は入力された複数のファイルから情報をマージし，ca850 のオプションで指定する「セクション・ファイル」を 1 つ出力します。

次図は，sf850 が出力するセクション・ファイルの例です。

図 9 － 2 sf850 が出力するセクション・ファイル例

```
//Created by sf850. at Thu Jan 22 17:26:25 2004
[tidata]
// [file:[func:]]variable  // section size total_freq Byte_freq Word_freq
"main.c:val1"  // data 4 10 10 0
"main.c:val2"  // data 4 8 8 0
"main.c:func1:val3" // -4 5 5 0
"i" // -4 3 3 0
"j" // -2 1 1 0
```

ファイルにおいて「//」以降はコメントとして扱われます。

セクション・ファイルにおける変数の表示は次のような形になっています。

```
[ セクション種別 ]
" ファイル名 : 関数名 : 変数名 " // コメント
" ファイル名 : 変数名 " // コメント
" 変数名 " // コメント
```

変数の表示には上のように 3 種類があり，変数の種類によります。その種類は次のようになります。

表 9 － 1 変数の種類と表示

表示	意味
ファイル名 : 関数名 : 変数名	関数内で宣言された静的変数。その関数名とファイル名も表示される。
ファイル名 : 変数名	ファイル内で宣言された静的変数。そのファイル名も表示される。
変数名	外部変数。変数名だけが表示される。

また，コメントは次のような形で出力されます。

```
section size total_freq Byte_frec Word_frec
```

それぞれの意味は次のようになります。

表 9 － 2 変数の表示とその意味

表示	意味
section	その変数の割り当てが明示的に指示されたセクション。 明示的に指示されていない変数に対しては “ - ” が表示されます。
size	変数のサイズ（バイト単位）サイズが不明な場合は 0 となります。
total_freq	変数の参照頻度。その変数に対するロード／ストア命令の出現回数を意味します。
Byte_freq	変数の参照頻度中，バイト単位で参照された回数を意味します。
Word_frec	変数の参照頻度中，ワード単位で参照された回数を意味します。

sf850 は，すべての変数を .tidata セクションへ配置するようなセクション・ファイルを出力します。.tidata セクションのメモリ・サイズは 256 バイトであるため，その範囲に収まらない場合は，使用する側の判断で修正する必要があります。

ただし，“ -O オプション ”を指定すると，利用頻度の順に，範囲に収まる分だけの変数を判断して出力するため，そのまま ca850 に対する入力として用いることができます。また，“ -O オプション ”を指定した場合，[tidata] の代わりに [tidata_word] ／ [tidata_byte] に分けて出力します。

次図は，“ -O オプション ” を指定したときに出力されるセクション・ファイルの例です。

図 9 － 3 sf850 が -O オプション指定で出力するセクション・ファイル例

```
// Created by sf850. at Thu Jan 22 17:26:25 2004
[tidata_byte]
// [file:[func:]]variable  // section size total_freq Byte_freq Word_freq
"a.c:si1"  // data 4 10 10 0
"a.c:si2"  // data 4 8 8 0
"a.c:f1:sfil" // - 4 5 5 0
"i"  // -4 3 3 0
"J"  // -2 1 1 0
[tidata_word]
"a.c:si3"  // data 4 10 0 10
"a.c:si4"  // data 4 8 0 8
"a.c:f1:sfi2" // -4 5 2 3
"1"  // -4 3 2 1
"m"  // -2 1 0 1
```

tidata 属性，tidata.word 属性，tidata.byte 属性のセクションに限らず，他の属性のセクションもセクション・ファイルには記述できます。

セクション種別に指定可能な文字列は次のとおりです。

表 9 － 3 ca850 で指定可能なセクション種別

種別指定文字列	割り当てられるセクション
tidata	初期値を設定したバイト・データは .tidata.byte セクション，初期値を設定したハーフ・ワード以上のデータは .tidata.word セクションに割り当てられます。 初期値を設定しないバイト・データは .tibss.byte セクション，初期値を設定しないハーフ・ワード以上のデータは .tibss.word に割り当てられます。
data	初期値を設定した場合，.data セクションに割り当てられ，設定しない場合，.bss セクションに割り当てられます。
sdata	初期値を設定した場合，.sdata セクションに割り当てられ，設定しない場合，.sbss セクションに割り当てられます。
sedata	初期値を設定した場合，.sedata セクションに割り当てられ，設定しない場合，.sebss セクションに割り当てられます。
sidata	初期値を設定した場合，.sidata セクションに割り当てられ，設定しない場合，.sibss セクションに割り当てられます。
const	.const セクションに割り当てられます。
sconst	.sconst セクションに割り当てられます。

[注意事項]

- セクション名を指定する [ ] 内で，セクション名の前後に空白を挿入できません。
  たとえば，[tidata] の場合，“ tidata ” の前後に空白を挿入できません。
- 変数名は "（ダブルクォーテーション）で囲んでください（V2.60 以前の書式も有効となります）。
- 変数は 1 行につき 1 つのみです。1 行に 2 つ以上となる修正，および 1 つの変数指定が複数行で示されるような修正はできません。
- ‘ : ’ の前後に空白を挿入できません。
- ファイル名にはパスを含めることができません。
- 関数や変数の定義がヘッダ・ファイル内にある場合，セクション・ファイルにおける “ ファイル名 ” は，ヘッダ・ファイル名ではなく，ヘッダ・ファイルをインクルードしている C 言語ソース・ファイル名となります。
- “ /* */ ”，または “ // ” 形式のコメントを挿入できます。
  ただし，セクション名や変数名をコメントで区切ることはできません。また，変数名の直後には空白が必要です。なお，コメントには，ASCII 文字，および EUC 日本語コードが使用できます。
- セクション・ファイルでセクション種別に “ data ” を指定した変数を，他のアセンブリ言語ソース・ファイルで参照する場合，data ／ bss 属性であることをアセンブラに通知するため，.option 疑似命令を使い “ .option data ” と指定して参照してください。また，“ sdata ” を指定した変数を，他のアセンブリ言語ソース・ファイルで参照する場合，sdata ／ sbss 属性であることをアセンブラに通知するため，.option 疑似命令を使い “ .option sdata ” と指定して参照してください。

記述例

```
// セクション・ファイル
[data]
"a.c:dat1"            // 初期値ありの場合，.data セクションとなる
"b.c:dat2"            // 初期値なしの場合，.bss セクションとなる
[sdata]
"a.c:sdat1"           // 初期値ありの場合，.sdata セクションとなる
"b.c:sdat2"           // 初期値なしの場合，.sbss セクションとなる

# アセンブリ言語ソース・ファイル
       .option data _dat1
       .text
       mov    $_dat1, r11   -- .data セクションとみなし，命令展開する
       .option data _dat2
       .text
       mov    $_dat2, r12   -- .bss セクションとみなし，命令展開する
       .option sdata _sdat1
       .text
       mov    $_sdat1, r13  -- .sdata セクションとみなし，命令展開しない
       .option sdata _sdat2
       .text
       mov    $_sdat2, r14  -- .sbss セクションとみなし，命令展開しない
```

9.3 操作方法

コマンド・ラインからの起動，および PM+ を使用したときのセクション・ファイルの利用方法を説明します。

9.3.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
sf850 [ オプション ]... ファイル名 [ ファイル名 ]...
       [ ]    ：[ ] 内は省略できます。
       ...    ：直前の [ ] 内のパターンの繰り返しができます。
```

9.3.2 PM+ による方法

[セクションファイルジェネレータオプションの設定] ダイアログは，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [セクションファイルジェネレータオプションの設定] を選択

セクション・ファイル・ジェネレータの起動は，プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

セクション・ファイル・ジェネレータが出力するセクション・ファイルのデフォルト名は，プロジェクト・ファイル名の拡張子を “ .sf ” に置き換えたものとなります。オプションで出力ファイル名を指定することができます。

ただし，PM+ 上からセクション・ファイル・ジェネレータを起動するとき表示される [セクションファイルジェネレータオプションの設定] ダイアログの [ファイル] タブで，“ 使用する ”がチェックされている必要があります。

9.3.3 コマンド入力による利用

この項では，コマンド・ライン上からセクション・ファイルを利用する方法について説明します。

(1) 始めに頻度情報ファイルを生成します。ca850 のオプション“ -Xcre_sec_data_only ”を指定して，C 言語ソース・ファイルをコンパイルすると，その C 言語ソース・ファイル中の外部変数，静的変数に関する頻度情報ファイルが生成されます。ファイル名はデフォルトで“ C 言語ソース・ファイル名 .sec ”です。
-Xcre_sec_data_only オプションと同時にファイル名を指定した場合は，指定したファイル名が頻度情報ファイル名になります。

例

```
ca850 -cpu 3201 -Xcre_sec_data_only=secsrc func1.c
```

この場合，func1.c に関する頻度情報をファイル“ secsrc ”に出力します。

(2) 生成した頻度情報ファイルを sf850 に入力し，セクション・ファイルを出力します。このとき，変数を tidata 属性，tidata.word 属性，tidata.byte 属性のセクションへ配置指定するセクション・ファイルが生成されます。

例

```
sf850 func1.sec func2.sec func3.sec -o secfile
```

この場合，頻度情報ファイル func1.sec，func2.sec，func3.sec をセクション・ファイルとして 1 つにまとめ，ファイル“ secfile ”に出力します。ファイル数が多い場合は，コマンド・ファイルを作っておくと便利です。コマンド・ファイルについては，「3.7.2 コマンド・ファイル」を参照してください。

(3) 出力されたセクション・ファイルは，デフォルトではすべての変数を .tidata 属性のセクションへ配置する指定になっているため，必要に応じてセクション・ファイルを修正します。
なお，sf850 起動時“ -O オプション ”を指定すると，参照頻度の高い順に .tidata 属性のセクションのメモリ範囲に収まる分の変数を自動的に選択できます。

(4) ca850 のオプション“ -Xsec_file ”を指定して，C 言語ソース・ファイルを再コンパイルします。コンパイルの結果，入力したセクション・ファイルに従ったセクション配置のオブジェクト・ファイルが生成されます。

例

```
ca850 -cpu 3201 -Xsec_file secfile func1.c func2.c func3.c
```

この場合，セクション・ファイルとして secfile を入力し，func1.c，func2.c，func3.c をコンパイルします。

9.3.4 PM+ による利用

この項では，PM+ を使用したときのセクション・ファイルの利用方法について説明します。

PM+ 上から操作するとき，[セクションファイルジェネレータオプションの設定] から表示される [セクションファイルジェネレータオプションの設定] ダイアログの [ファイル] タブで，“ 使用する ” をチェックします。これによって ca850 は，頻度情報ファイルを作成後，自動的に sf850 を起動してセクション・ファイルを生成し，さらにそのセクション・ファイルを使用してコンパイルします。

なお，上記の方法を使用すると，出力されるセクション・ファイルは，すべての変数を .tidata 属性のセクションへ配置する指定になります。セクション・ファイルを修正して使用したい場合は，“ 使用する ” にチェックせず，次の方法によって行う必要があります。

(1) 始めに頻度情報ファイルを生成します。[コンパイラオプションの設定] ダイアログの [出力ファイル] タブにおける “ 頻度情報ファイル [-Xcre_sec_data] ” をチェックします。この際，エディット・ボックスにファイル名を指定すると，頻度情報を出力するファイル名を指定できます。

なお，頻度情報ファイルは，C 言語ソース・ファイル別に作成する必要があります。上の設定のように，出力ファイル名を指定し，かつ，すべての C 言語ソース・ファイルに対する設定であった場合（ソース個別に設定したものではない場合），1 つ 1 つの C 言語ソース・ファイルをコンパイルするたびに，頻度情報が func1.sec に次々に上書きされてしまいます。つまり，最終的に完成する func1.sec は最後にコンパイルした C 言語ソース・ファイルに対する情報だけになります。

ただし，ファイル名を指定しなかった場合，頻度情報ファイルは「C 言語ソース・ファイル名 .sec」というファイル名になり，C 言語ソース・ファイルごとに頻度情報ファイルが生成されます。つまり，すべての頻度情報ファイルを作成するときは，特にファイル名を指定したい場合は C 言語ソース・ファイル個別のオプションとして指定し，そうでない場合は，ファイル名を指定しない方がよいです。

(2) 生成した頻度情報ファイルを sf850 に入力し，セクション・ファイルを出力します。sf850 の起動はコマンド・ライン上から行います。このとき，変数を tidata 属性，tidata.word 属性，tidata.byte 属性のセクションへ配置指定するセクション・ファイルが生成されます。

例

```
sf850 func1.sec func2.sec func3.sec -o secfile
```

この場合，頻度情報ファイル func1.sec，func2.sec，func3.sec をセクション・ファイルとして 1 つにまとめ，ファイル “ secfile ” に出力します。ファイル数が多い場合は，コマンド・ファイルを作っておくと便利です。コマンド・ファイルについては，「3.7.2 コマンド・ファイル」を参照してください。

(3) 出力されたセクション・ファイルは，デフォルトではすべての変数を .tidata 属性のセクションへ配置する指定になっているため，必要に応じてセクション・ファイルを修正します。

なお，sf850 起動時に “ -O オプション ” を指定すると，参照頻度の高い順に .tidata 属性のセクションのメモリ範囲に収まる分の変数を自動的に選択できます。

(4) [コンパイラオプションの設定] ダイアログの [入力ファイル] タブにおける “ セクションファイル [-Xsec_file] ” のエディット・ボックスに生成したセクション・ファイルの名前を指定し，リビルドします。ビルドの結果，入力したセクション・ファイルに従ったセクション配置のオブジェクト・ファイルが生成されます。

9.4 オプションの種類と機能

次に sf850 のオプションを示します。

〔オプション説明でのマーク〕

【PM+】	PM+ で指定項目が存在するオプション

9.4.1 オプション

sf850 の通常のオプションを設定します。

`-o`

【PM+】

利用頻度の高い順に，.tidata セクションに配置可能な分だけの変数を判断して出力します。.tidata セクションに配置可能なサイズは 256 バイトであり，内部的にはバイト・データの .tidata.byte（128 バイト）とワード・データの .tidata.word に分かれています。このオプションの指定では，その合計が 256 バイトになるまで変数を選択し，セクション・ファイルに出力します。

ただし，バイト・データは 128 バイトに達したところで選択を終了します。このオプションを省略した場合，出現したすべての変数をセクション・ファイルに出力します。

`-v`

sf850 のバージョン情報を標準出力に出力し，終了します。

`-Xcs[=name]`

【PM+】

-O オプション指定時に *name* で指定されたセクションに割り付けられている変数を最適化の対象にしません。*name* はリンク・ディレクティブ・ファイルに指定するセクション名を指定します。

ただし，bss 属性のセクションは .bss ／ .sbss の部分を .data ／ .sdata に置き換えてください。*name* を省略した場合は，すべてのセクション名が指定されたものとみなします。*name* に .tidata が指定された場合は .tidata.word，.tidata.byte の 2 つが指定されたものとみなします。

`-Xcv=name`

【PM+】

-O オプション指定時に *name* で指定された変数を最適化の対象にしません。*name* は表 9 － 1 の表示と同じ書式で指定します。

`-cl num`

【PM+】

出力するセクション・ファイルのコメント・レベルを指定します。*num* には次のものが指定できます。

0	コメントを出力しない。
1	日付などのファイル生成情報，および変数情報とその説明を出力する。変数情報は，セクション名，サイズ，利用頻度である外部変数でセクション名が決定されていない場合，‘ - ’を出力する。
2	レベル `1` に加え，書式ガイドを出力する `-O` 指定時は，`.tidata` セクションに入らないと判断された変数もコメントとして出力する。

このオプションを省略した場合，コメント・レベル 1 とみなします。

+err_file=*file*

エラー・メッセージをファイル *file* に追加保存します。

-err_file=*file*

エラー・メッセージをファイル *file* に上書き保存します。

-h

-help

sf850 **のオプションの説明を標準出力し，終了します。**

-ns

【PM+】

出力するセクション・ファイル中の変数名をソートせず，出現した順番でソートします。このオプションを省略した場合，利用頻度順でソートします。利用頻度が同じ場合，サイズの小さい順にソートします。

-o *name*

【PM+】

name **を出力するセクション・ファイル名とします。このオプションを省略した場合，標準出力に出力します。**

-size_tidata=*num*

【PM+】

-O **オプション指定時に** .tidata.word ／ .tidata.byte **セクションに変数を配置するサイズの上限を** *num* **バイトに制限します。**

-size_tidata_byte=*num*

【PM+】

-O **オプション指定時に** .tidata.byte **セクションに変数を配置するサイズの上限を** *num* **バイトに制限します。**

-sname

【PM+】

出力するセクション・ファイル中の変数名を変数名の辞書順にソートします。変数名が同じ場合，ファイル名，関数名の辞書順にソートします。

-ssection

【PM+】

出力するセクション・ファイル中の変数名を割り当てられるセクション名の辞書順にソートします。
セクション名が同じ場合，利用頻度の高い順にソートします。

-ssize

【PM+】

出力するセクション・ファイル中の変数名をサイズの小さい順にソートします。サイズが同じ場合，利用頻度の高い順にソートします。

-v

【PM+】

sf850 **の実行過程を表示します。**

@cfile

【PM+】

cfile をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

[注意事項]

これらのオプションの中には，他のオプションと同時に指定された場合，無効になるものがあります。

- ソートに関するオプション，-o，および -cl は，複数指定された場合，あとから指定されたものを有効とし，その他を無効とします。
- -V，-h，-help が同時に指定された場合，先に指定されたものを有効とし，その他を無効とします。
- -O とソートに関するオプションが同時に指定された場合，-O を有効とし，ソートに関するオプションは無効とします。
- sf850 に入力する頻度情報ファイルは CA850 が出力したそのままの状態のものを使用してください。その内容を修正した頻度情報ファイルを入力した場合，動作は保証されません。

sf850 によって出力されるセクション・ファイルの内容については「9.2 セクション・ファイルの書式」を参照してください。

9.5 PM+ での設定

この節では，対象プロジェクトのソース・ファイルに対して，sf850 のコマンド・オプションを設定するダイアログについて説明します。

9.5.1 [セクションファイルジェネレータオプションの設定] ダイアログ

[セクションファイルジェネレータオプションの設定] ダイアログの上部には，次の 3 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

表 9 － 4 [セクションファイルジェネレータオプションの設定] ダイアログ

タブ	内容
[ファイル]	ファイルの設定
[オプション]	オプションの設定
[その他]	その他の設定

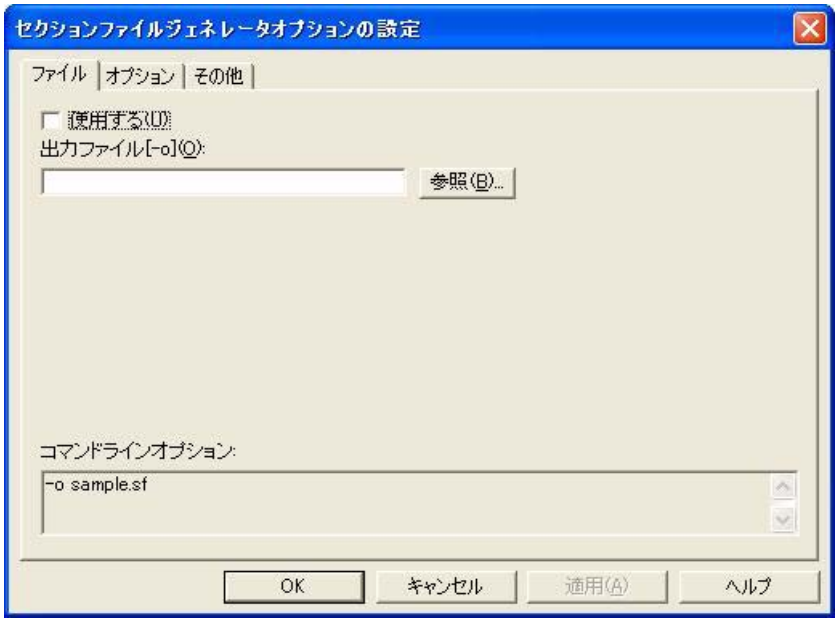
なお，[セクションファイルジェネレータオプションの設定] ダイアログには，コマンド・ラインから起動する場合のオプション名が “ [ ] ” に表示されています。

注 セクション・ファイル・ジェネレータのオプションを変更すると，生成されるファイルは削除されます。編集して使用する場合には，別フォルダにコピーしてから使用してください。

[ファイル]

セクション・ファイル・ジェネレータのファイルの設定を行います。

図９－４ [セクションファイルジェネレータオプションの設定] ダイアログ：[ファイル] タブ



(1) 使用する

ビルド時にセクション・ファイル・ジェネレータを使用するかどうかを指定するチェック・ボックスです。チェックされている場合は，セクション・ファイル・ジェネレータを使用します。使用すると，すべての C 言語ソース・ファイルに対して頻度情報ファイル“ .sec ”が，プロジェクト・フォルダ，または中間出力用フォルダに作成されます。

(2) 出力ファイル [-o]

出力するセクション・ファイル名（拡張子は“ .sf ”）を指定します。ファイル名に空白は使用できません。ファイル名を指定すると，-o オプションを指定したのと同じになります。ファイル名を指定しなかった場合，プロジェクト・ファイル名の拡張子を“ .sf ”に変えたファイル名が指定されたものとみなします。また，[参照] ボタンにより表示されるダイアログで，ファイルを選択することができます。

(3) コマンドラインオプション

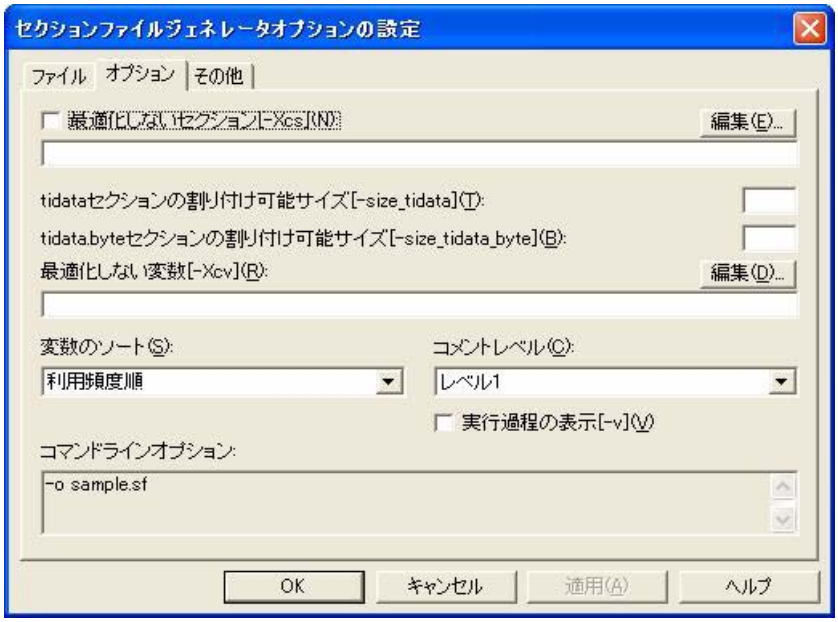
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[オプション]

セクション・ファイル・ジェネレータのオプションの設定を行います。

図９－５ [セクションファイルジェネレータオプションの設定] ダイアログ：[オプション] タブ



(1) 最適化しないセクション [-Xcs]

“ 変数のソート ”における“ 最適配置 [-O] ”を指定時にセクションに割り付けられている変数を最適化の対象にしません。チェックされている場合は，すべてのセクションが指定されたものとみなします。エディット・ボックスにセクションが指定されている場合は，そのセクションに割り付けられている変数を最適化の対象にしません。

なお，複数のセクションを指定する場合は“；（セミコロン）”で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，セクションをダイアログ上で編集することができます。

(2) tidata セクションの割り付け可能サイズ [-size_tidata]

“ 変数のソート ”における“ 最適配置 [-O] ”を指定時に tidata.word ／ tidata.byte セクションに変数を配置するサイズの上限を制限します。サイズは，10 進数で指定します。

(3) tidata.byte セクションの割り付け可能サイズ [-size_tidata_byte]

“ 変数のソート ”における“ 最適配置 [-O] ”を指定時に tidata.byte セクションに変数を配置するサイズの上限を制限します。サイズは，10 進数で指定します。

(4) 最適化しない変数 [-Xcv]

“ 変数のソート ”における“ 最適配置 [-O] ”を指定時に変数を最適化の対象にしません。

なお，複数の変数を指定する場合は“；（セミコロン）”で区切ります。[編集] ボタンの選択により，[オプションの編集] ダイアログを表示し，変数をダイアログ上で編集することができます。

(5) 変数のソート

変数のソートに関するオプションを設定します。

(a) **ソートしない** [-ns]

出力するセクション・ファイル中の変数名をソートせず，現れた順にソートします。

(b) **利用頻度順**

利用頻度の高い順にソートします（デフォルト）。利用頻度が同じ場合，サイズの小さい順にソートします。

(c) **変数名** [-sname]

出力するセクション・ファイル中の変数名を変数名の辞書順にソートします。変数名が同じ場合，ファイル名，関数名の辞書順にソートします。

(d) **セクション名** [-ssection]

出力するセクション・ファイル中の変数名を割り当てられているセクション名の辞書順にソートします。セクション名が同じ場合，利用頻度の高い順にソートします。

(e) **変数のサイズ** [-ssize]

出力するセクション・ファイル中の変数名をサイズの小さい順にソートします。サイズが同じ場合，利用頻度の高い順にソートします。

(f) **最適配置** [-O]

利用頻度の高い順に，.tidata.byte **セクション／** .tidata.word **セクションに配置可能な分だけの変数を判断して出力します。**.tidata **セクションに割り当て可能なサイズは** 256 **バイトであり，内部的にはバイト・データの** .tidata.byte**（**128 **バイト）とワード・データの** .tidata.word **に分かれています。**8 **ビット・データは，**.tidata.byte **の領域にのみ配置可能です。**

セクション・ファイル・ジェネレータはその合計が 256 **バイトになるまで変数を選択しセクション・ファイルに出力します。ただし，**8 **ビット・データは** 128 **バイトに達したところで選択を終了します。**

(6) コメントレベル

出力するセクション・ファイルのコメント・レベルを指定します。

(a) **出力しない** [-cl0]

出力しません。

(b) **レベル** 1

日付などのファイル生成情報，および変数情報とその説明を出力します（デフォルト）。変数情報は，セクション名，サイズ，利用頻度です。外部変数でセクション名が決定されていない場合，‘ - ’ を出力します。

(c) **レベル** 2[-cl2]

レベル 1 **に加え，書式ガイドを出力します。[最適配置] 指定時は，**.tidata **セクションに入らないと判断された変数もコメントとして出力します。**

(7) 実行過程の表示 [-v]（V）

セクション・ファイル・ジェネレータの実行状況の詳細をアウトプット・ウインドウに表示します。

(8) コマンドラインオプション

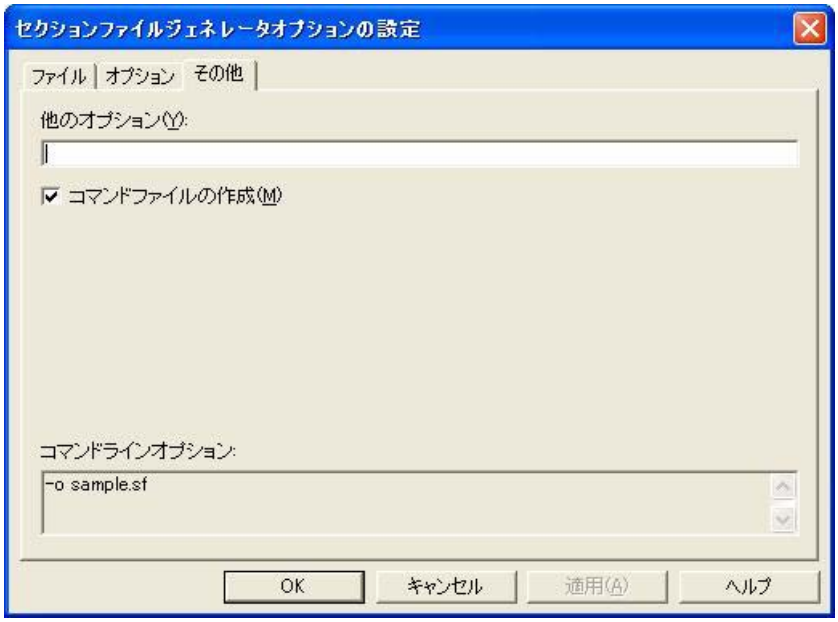
ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[その他]

セクション・ファイル・ジェネレータのその他の設定を行います。

図９－６ [セクションファイルジェネレータオプションの設定] ダイアログ：[その他] タブ



(1) 他のオプション

前記“ セクションファイルジェネレータの設定 ”では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

なお，セクション・ファイル・ジェネレータに関するオプションは [セクションファイルジェネレータの設定] ダイアログですべて指定できるため，他のオプションを使用する必要はありません。

(2) コマンドファイルの作成

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。チェックされている場合は，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，文字列の制限を意識する必要がなくなります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

なお，このオプションはデフォルトではチェックされていません。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

(3) コマンドラインオプション

ダイアログで設定したプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

第 10 章　ダンプ・コマンド

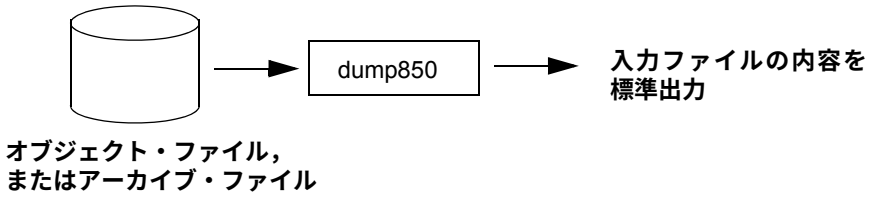
この章では，ダンプ・コマンド（dump850）の概要，操作方法，表示形式について説明します。

10.1　ダンプ・コマンドとは

ダンプ・コマンドとは，指定されたオブジェクト・ファイルやアーカイブ・ファイルの内容や情報を表示します。作成したオブジェクト・ファイルやアーカイブ・ファイル中の，セクション／セグメントのアドレスや属性，シンボル名などの情報を確認する場合などに利用します。

C コンパイラ・パッケージに入っている“ dump850 ”がダンプ・コマンドです。

図 10 － 1　dump850 における動作の流れ



なお，アーカイブ・ファイルをダンプ・コマンドに入力した場合，アーカイブ・ファイル内に“ オブジェクト・ファイルではないメンバ ”が存在した場合，警告メッセージを出力して，次のメンバの処理を行います。ただし，-e オプションを指定されている場合を除きます。

オプションについての詳細は「10.3　オプションの種類と機能」を参照してください。

10.2 操作方法

この節では，dump850 コマンドの操作方法について説明します。

10.2.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
dump850 [ オプション ]... ファイル名 [ ファイル名 ]...
        [ ]    ：[ ] 内は省略できます。
        ...    ：直前の [ ] 内のパターンの繰り返しができます。
```

10.2.2 PM+ による方法

ダンプ・コマンド起動用のダイアログ（[オブジェクト解析ツール] ダイアログ）は，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [オブジェクト解析ツールの起動] を選択したのち，[ダンプ] タブを選択

ダンプ・コマンドの起動は，プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

10.3 オプションの種類と機能

次に，dump850 のオプションを示します。

-A

指定されたオブジェクト・ファイル，またはアーカイブ・ファイルのすべての内容を表示します。このオプションは“-a b c f g h l k l m r s t”を指定したことに等しくなります。どのオプションも指定されなかった場合，-A オプションが指定されたものとみなします。

-T

アーカイブ・ヘッダの内容の表示においてメンバの更新年月日を表示しません。

-V

dump850 のバージョン情報を標準出力に出力し，終了します。

-a

指定されたアーカイブ・ファイル内に存在する，すべてのメンバのアーカイブ・ヘッダの内容を表示します。

-b

デバッグ情報の内容を表示します。

-c

ストリング・テーブルの内容を表示します。

-d *num*

セクション・ヘッダ・テーブル・インデックス *num* で示されるセクション“ から ”表示します。

+d *num*

セクション・ヘッダ・テーブル・インデックス *num* で示されるセクション“ まで ”表示します。

-e

指定されたアーカイブ・ファイル内に存在する（アーカイブ・シンボル・テーブル，アーカイブ・ストリング・テーブル，およびオブジェクト・ファイル以外の）メンバの内容を表示します。

-f

指定されたオブジェクト・ファイル，またはアーカイブ・ファイル内に存在するすべてのメンバの ELF ヘッダの内容を表示します。

-g

指定されたアーカイブ・ファイルのアーカイブ・シンボル・テーブル内に存在する外部シンボルの内容を表示します。

-h

指定されたオブジェクト・ファイル，またはアーカイブ・ファイル内に存在するすべてのセクション・ヘッダの内容を表示します。

-i

指定されたオブジェクト・ファイル，またはアーカイブ・ファイル内に存在するすべてのプログラム・ヘッダの内容を表示します。

-k

グローバル・ポインタ・テーブルの内容を表示します。

-l

ライン・ナンバ情報の内容を表示します。

-m

指定されたアーカイブ・ファイルのアーカイブ・ストリング・テーブル内に存在するストリングの内容を表示します。

-n *name*

セクション名 *name* で示されるセクションの内容を表示します。

-p

タイトルを表示しません。

-r

リロケーション情報の内容を表示します。

-s

セクションの内容を表示します。

-t [*num*]

シンボル・テーブルの内容を *num* 番目のシンボル・テーブル・エントリ“から”表示します。*num* を省略した場合，1 番目のシンボル・テーブル・エントリから表示します。

+t *num*

シンボル・テーブルの内容を *num* 番目のシンボル・テーブル・エントリ“まで”表示します。

-v

セクション属性などの値を数字ではなく，その値の意味を示す文字列で表示します（「10.5.2 要素の値と意味」参照）。

-z *name* [*num*]

関数 *name* のライン・ナンバ情報の内容を *num* 番目のライン・ナンバ・エントリから表示します。*num* を省略した場合，1 番目のライン・ナンバ・エントリから表示します。

@cfile

cfile をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

10.4 PM+ での設定

この節では，対象プロジェクトのファイルに対して，dump850 のコマンド・オプションを設定するダイアログについて説明します。

10.4.1 [オブジェクト解析ツール] ダイアログ

[オブジェクト解析ツール] ダイアログの上部には，次の 3 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

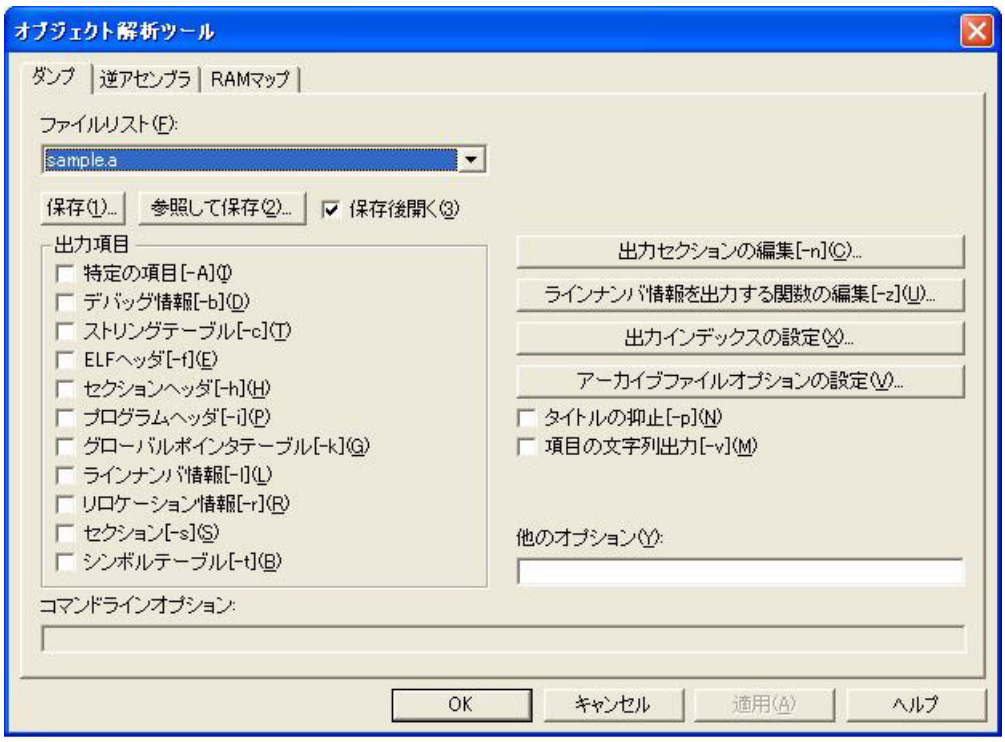
表 10 － 1 [オブジェクト解析ツール] ダイアログ（dump850）

タブ	内容
[ダンプ]	ダンプ・コマンドに関するオプションを設定
[逆アセンブラ]	ディスアセンブラに関するオプションを設定
[RAM マップ]	RAM マップに関するオプションを設定

[ダンプ]

ダンプ・コマンド起動用ダイアログは，プロジェクト・マネージャでプロジェクトを設定後，[ツール] メニュー→ [オブジェクト解析ツールの起動] でオープンするダイアログの [ダンプ] タブを選択します。

図 10 － 2 [オブジェクト解析ツール] ダイアログ：[ダンプ] タブ



(1) ファイルリスト

プロジェクトでビルド対象となるオブジェクト・ファイルをドロップダウン・リストに表示します（リストにはパスは表示されません）。

リストとして次のファイルが対象となります。

- ROM 化プロセッサの出力ファイル（実行可能オブジェクト・ファイルをビルドするプロジェクトで ROM 化を指定している場合のみ）
- リンカの出力ファイル（実行可能オブジェクト・ファイルをビルドするプロジェクトのみ）
- アーカイバの出力ファイル（ライブラリを出力するプロジェクトのみ）
- コンパイラの出力ファイル
- アセンブラの出力ファイル

[保存] ボタンにより，[名前を付けて保存] ダイアログを開き，“ ファイルリスト ”で指定したオブジェクト・ファイルのダンプ結果を保存します。デフォルトでは，“ ファイルリスト ” で指定したオブジェクト・ファイルが存在するフォルダ内に，“ オブジェクト・ファイル名 .dmp ” をファイル名として保存します。

なお，[参照して保存] ボタンにより，オブジェクト・ファイルを指定してダンプ結果を保存することができます。この場合，[ファイルを開く] ダイアログが開き，オブジェクト・ファイルを指定します。デフォルトで

は，前回の [ファイルを開く] ダイアログで指定したフォルダ，またはプロジェクト・フォルダが表示されます。

[ファイルを開く] ダイアログでオブジェクト・ファイルを指定すると，指定したオブジェクト・ファイルに対するダンプ結果を [保存] ボタンの選択と同様の形式で保存します。

(2) 保存後開く

[保存] ／ [参照して保存] ボタンによりダンプ結果を保存したのち，PM+ 上で設定したエディタで保存したファイルを開くがどうかを指定します。デフォルトでチェックされています。

(3) 出力項目

ダンプ結果として出力する項目をチェック・ボックスにより選択します。

(a) 特定の項目

-A オプションで指定する，特定の項目を出力します。

チェックすることにより，-A オプションを指定した場合と同等となります。

(b) デバッグ情報

-B オプションで指定する，デバッグ情報を出力します。

チェックすることにより，-B オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(c) ストリングテーブル

-C オプションで指定する，ストリングテーブルを出力します。

チェックすることにより，-C オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(d) ELF ヘッダ

-f オプションで指定する，ELF ヘッダを出力します。

チェックすることにより，-f オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(e) セクションヘッダ

-h オプションで指定する，セクション・ヘッダを出力します。

チェックすることにより，-h オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(f) プログラムヘッダ

-i オプションで指定する，プログラム・ヘッダを出力します。

チェックすることにより，-i オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(g) グローバルポインタテーブル

-k オプションで指定する，グローバルポインタテーブルを出力します。

チェックすることにより，-k オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(h) ラインナンバ情報

-l オプションで指定する，ラインナンバ情報を出力します。

チェックすることにより，-l オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(i) リロケーション情報

-r オプションで指定する，リロケーション情報を出力します。

チェックすることにより，-r オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(j) セクション

-s オプションで指定する，セクションを出力します。

チェックすることにより，-s オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(k) シンボルテーブル

-t オプションで指定する，シンボルテーブルを出力します。

チェックすることにより，-t オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(4) タイトルの抑止

-p オプションで指定する，タイトルの出力を抑止するかどうかを指定します。

チェックすることにより，-p オプションを指定した場合と同等になります。

(5) 項目の文字列出力

-v オプションで指定する，一部の項目を文字列で出力するかどうかを指定します。

チェックすることにより，-v オプションを指定した場合と同等になります。

(6) 他のオプション

前記の“ dump850 コマンドのオプション ”では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

なお，dump850 コマンドに関するオプションは [オブジェクト解析ツール] ダイアログですべて指定できるため，他のオプションを使用する必要はありません。

(7) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

[ボタン]

(1) [出力セクションの編集] ボタン

-n オプションで指定する，出力セクションを編集するための [オプションの編集] ダイアログを開きます。

(2) [ラインナンバ情報を出力する関数の編集] ボタン

-z オプションで出力する，ラインナンバ情報を出力する関数を編集するための [オプションの編集] ダイアログを開きます。

(3) [出力インデックスの設定] ボタン

出力インデックスに関する設定を行うために，[出力インデックスの設定] ダイアログを開きます。

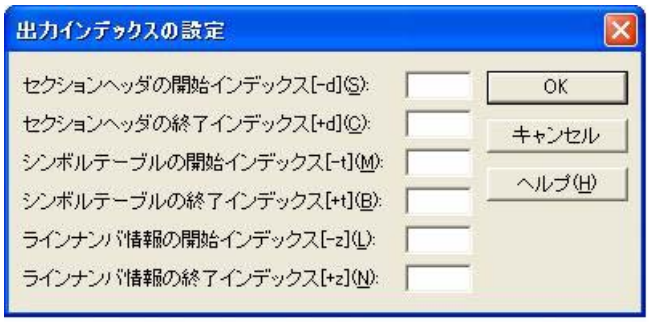
(4) [アーカイブファイルオプションの設定] ボタン

アーカイブ・ファイルに対するオプションの設定を行うために，[アーカイブファイルオプションの設定] ダイアログを開きます。

10.4.2 [出力インデックスの設定] ダイアログ

出力インデックスに関する設定を行うためのダイアログです。
それぞれのエディット・ボックスにおいて，10 進／ 16 進で指定します。

図 10 － 3 [出力インデックスの設定] ダイアログ



(1) セクションヘッダの開始インデックス

-d オプションで指定する，セクションヘッダの開始インデックスを設定します。
指定することにより，-d オプションを指定した場合と同等となります。

(2) セクションヘッダの終了インデックス

+d オプションで指定する，セクションヘッダの終了インデックスを設定します。
指定することにより，+d オプションを指定した場合と同等となります。

(3) シンボルテーブルの開始インデックス

-t オプションで指定する，シンボルテーブルの開始インデックスを設定します。
指定することにより，-t オプションを指定した場合と同等となります。

(4) シンボルテーブルの終了インデックス

+t オプションで指定する，シンボルテーブルの終了インデックスを設定します。
指定することにより，+t オプションを指定した場合と同等となります。

(5) ラインナンバ情報の開始インデックス

-z オプションで指定する，ラインナンバ情報の開始インデックスを設定します。
指定することにより，-z オプションを指定した場合と同等となります。

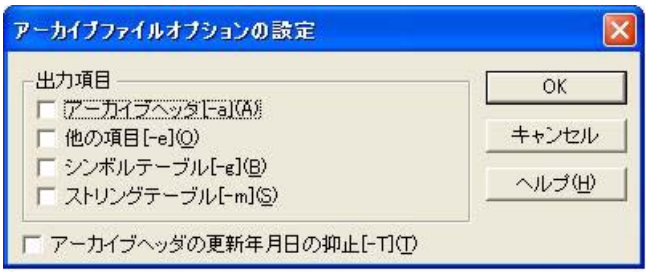
(6) ラインナンバ情報の終了インデックス

+z オプションで指定する，ラインナンバ情報の終了インデックスを設定します。
指定することにより，+z オプションを指定した場合と同等となります。

10.4.3 [アーカイブファイルオプションの設定] ダイアログ

アーカイブ・ファイルに対するオプションの設定を行うためのダイアログです。

図 10 － 4 [アーカイブファイルオプションの設定] ダイアログ



(1) 出力項目

出力項目を次のチェック・ボックスにより選択します。

(a) アーカイブヘッダ

-a オプションで指定する，アーカイブ・ヘッダを出力します。

チェックすることにより，-a オプションを指定した場合と同等となります。

(b) 他の項目

-e オプションで指定する，アーカイブ・ファイルの他の項目を出力します。

チェックすることにより，-e オプションを指定した場合と同等となります。

(c) シンボルテーブル

-g オプションで指定する，アーカイブ・ファイルのシンボルテーブルを出力します。

チェックすることにより，-g オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(d) ストリングテーブル

-m オプションで指定する，アーカイブ・ファイルのストリングテーブルを出力します。

チェックすることにより，-m オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(2) アーカイブヘッダの更新年月日の抑止

-T オプションで指定する，アーカイブ・ヘッダの更新年月日の出力を抑止します。

チェックすることにより，-T オプションを指定した場合と同等となります。

10.5 出力形式

この節では，dump850 コマンドの出力形式について説明します。

10.5.1 ダンプ・リストの表示内容

(1) アーカイブ・ヘッダ

アーカイブ・ヘッダの内容を表示します。

```
                        ***ARCHIVE HEADER***

Date (a)      Uid (b)       Gid (c)       Mode (d)      Size (e)      Member Name (f)
0x3158DE73    0             0             0100664       0x2B8         atof.o
```

(a) メンバの更新年月日

(b) ユーザ ID

(c) グループ ID

(d) ファイルのパーミッション

(e) メンバの総バイト数

(f) メンバ名

(2) アーカイブ・シンボル・テーブル

アーカイブ・シンボル・テーブルの内容を表示します。

```
                      ***ARCHIVE SYMBOL TABLE***

                    Offset (a)          Name (b)
                    0x1f3c              _abs
```

(a) シンボルを含んでいるメンバへのファイル内オフセット

(b) シンボル名

(3) アーカイブ・ストリング・テーブル

アーカイブ・シンボル・テーブルの内容を表示します。

```
                      ***ARCHIVE STRING TABLE***

                   Offset (a)          Name (b)
                   0x1100              foo.o
```

(a) オフセット

(b) メンバ名

(4) ELF ヘッダ

ELF ヘッダの内容を表示します。

```
***ELF HEADER***

Class (a)       Data (b)        Type (c)        Machine (d)     Version (e)
Entry (f)       Phoff (g)       Shoff (h)       Flags (i)       Ehsize (j)
Phentsize (k)   Phnum (l)       Shentsz (m)     Shnum (n)       Shstrndx (o)
1               1               1               070377          1
0x0             0x0             0x2A4           0x84            0x34
0x20            0               0x28            6               5
```

(a) クラス

(b) バイト・オーダ

(c) 種類

(d) プロセッサ

(e) 版番号

(f) エントリ・ポイント・アドレス

(g) プログラム・ヘッダ・テーブルのファイル内オフセット

(h) セクション・ヘッダ・テーブルのファイル内オフセット

(i) フラグ

(j) ELF ヘッダの数

(k) プログラム・ヘッダ・テーブルのエントリ・サイズ

(l) プログラム・ヘッダ・テーブルのエントリの数

(m) セクション・ヘッダ・テーブルのエントリのサイズ

(n) セクション・ヘッダ・テーブルのエントリの数

(o) セクション名を保持しているストリング・テーブルのセクション・ヘッダ・テーブル・インデックス

(5) プログラム・ヘッダ・テーブル

プログラム・ヘッダ・テーブルの内容を表示します。

```
***PROGRAM HEADER***

No. (a)          Type (b)         Offset (c)       Vaddr (d)        Paddr (e)
                 Filesz (f)       Memsz (g)        Flags (h)        Align (i)
1.               0                0x0              0x0              0x0
                 0x0              0x0              0x0              0x0
```

(a) インデックス
(b) セグメント・タイプ
(c) ファイル内オフセット
(d) 仮想アドレス
(e) 物理アドレス
(f) ファイル・サイズ
(g) メモリ・サイズ
(h) セグメント属性
(i) 整列条件

(6) セクション・ヘッダ・テーブル

セクション・ヘッダ・テーブルの内容を表示します。

```
***SECTION HEADER***

No. (a)     Type (b)      Flags (c)     Addr (d)      Offset (e)    Size (f)     Name (g)
            Link (h)      Info (i)      Adralgn (j)   Entsize (k)
1.          0x1           0x6           0x0           0x1           0x7556       .text
            0x0           0x0           0x4           0x0
```

(a) インデックス
(b) セクション・タイプ
(c) セクション属性
(d) 先頭アドレス
(e) ファイル内オフセット
(f) サイズ
(g) セクション名
(h) セクション・ヘッダ・テーブル・インデックス・リンク
(i) 情報
(j) 整列条件
(k) エントリのサイズ

(7) ストリング・テーブル

ストリング・テーブルの内容を表示します。

```
***STRING TABLE INFORMATION***

Index (a)          String (b)
0x1                .text
```

(a) インデックス

(b) 文字列

(8) シンボル・テーブル

シンボル・テーブルの内容を表示します。

```
***SYMBOL TABLE INFORMATION***

No.     Value   Size    Bind    Type    Other   Shndx   Name
1.      0x0     0x0     0       3       0       0x1     .text
(a)     (b)     (c)     (d)     (e)     (f)     (g)     (h)
```

(a) インデックス

(b) 値

(c) サイズ

(d) バインディング・クラス

(e) タイプ

(f) アザー

(g) セクション・ヘッダ・テーブル・インデックス

(h) シンボル名

(9) リロケーション情報

リロケーション情報（リロケーション・エントリの並び）の内容を表示します。

```
***SYMBOL TABLE INFORMATION***

Offset          Sym             Type            Addend
0x20            6               0x23            0x0
(a)             (b)             (c)             (d)
```

(a) オフセット

(b) シンボル・デーブル・インデックス

(c) リロケーション・タイプ

(d) 加数定数

(10) レジスタ・モード情報

レジスタ・モード情報の内容を表示します。

```
***REGISTER MODE INFORMATION***

SymIdx          TmpReg          ParReg
0x1             0x5             0x5
(a)             (b)             (c)
```

(a) シンボル・テーブル・インデックス

(b) 作業用レジスタの数

(c) レジスタ変数用レジスタの数

(11) グローバル・ポインタ・テーブル

グローバル・ポインタ・テーブルの内容を表示します。

```
***GPTAB INFORMATION***

Gnum            Gsize
0x4             0xc
(a)             (b)
```

(a) -G*num* の *num* ／データのサイズ

(b) 0 ／ワードで整列した場合のサイズ

(12) ライン・ナンバ情報

ライン・ナンバ情報の内容を表示します。

```
***LINE NUMBER INFORMATION***

Bfunc   Maddr   Daddr   Pad     Function Name   Num     Snum    Offset  Flags
0x0     0xA2    0xE28   0x0     _main           0x5     0x0     0x0     0x1
(a)     (b)     (c)     (d)     (e)             (f)     (g)     (h)     (i)
```

(a) サブセクションの始まり

(b) 関数のアドレス

(c) デバッグ情報のアドレス

(d) パディング

(e) 関数名

(f) 行番号

(g) 文の位置

(h) オフセット

(i) フラグ

(13) デバッグ情報

デバッグ情報の内容を表示します。

```
***DEBUG INFORMATION***

Tag             Attr            Aux
0x0016
size            0x00000026
                0x000c          0x00000E1C
(a)             (b)             (c)
```

(a) タグ

(b) 属性

(c) 補助情報

(14) PROGBITS データ

PROGBITS データの内容を表示します。

```
***PROGBITS DATA in HEX***

0x00000000 : 40 0E 00 00 21 2E 00 00 ...
```

セクション・タイプ PROGBITS のセクションのロウ・データの内容を 16 進数で表示します。

10.5.2 要素の値と意味

-v オプションを指定した場合，次の情報では，一部の要素に対し，数値ではなく，その値の示す文字列で意味を表示します。

- ELF ヘッダ
- プログラム・ヘッダ・テーブル
- セクション・ヘッダ・テーブル
- シンボル・テーブル
- リロケーション情報
- デバッグ情報

次に，-v 指定で文字列表示となる要素のうち，特徴的な要素について，値[注]，-v 指定時の表示，その意味を示します。

注 dump850 の出力する進数のままで値を示します。

(1) ELF ヘッダの「Flags」

値	-v 指定時の表示	意味
0x1	L__________	.vline セクションが存在する。
0x2	_D________	.vdebug セクションが存在する。
0x4	__P_______	PIC（Position Independent Code）オブジェクトである。
0x10	___R______	22，または 26 レジスタ・モードである。
0x20	____d_____	異なるレジスタ・モードが混在する。
0x40	_____r____	romp850 が出力したオブジェクトである。
0x80	______N___	デフォルトの関数呼び出し仕様である（旧仕様の呼び出しではない）。
0x100	_______M__	マスク・レジスタ機能を使用している。
0x200	________U_	プロローグ／エピローグ・ランタイムの callt 呼び出しコードが出力される可能性がある。
0x400	_________S	プロローグ／エピローグ・ランタイムの callt 呼び出し向けに CTBP が設定されている。

(2) プログラム・ヘッダ・テーブル「Type」

値	-v 指定時の表示	意味
1	Load	メモリにロードされるセグメント。
4	Note	補助的な情報を入れたセグメント。

(3) セクション・ヘッダ・テーブル「Type」

値	-v 指定時の表示	意味
0x1	Progbits	オブジェクト・ファイル内に実際の値を持っているもの（機械語命令や初期値ありデータ）に対するセクション。
0x2	Symtab	シンボル・テーブル。
0x3	Strtab	ストリング・テーブル。
0x4	Rela	リロケーション情報。
0x8	Nobits	オブジェクト・ファイル内に実際の値を持っていないもの（初期値なしデータ）に対するセクション。
0x9	Rel	リロケーション情報。
0x70000000	Gptab	グローバル・ポインタ・テーブル（最初のエントリは ca，および as850 に対して指定された -G*num* の *num* と 0，2 番目以降のエントリはデータのサイズとワードで整列した場合のサイズ）。
0x70000001	Regmode	レジスタ・モード機能を用いて生成した，リンク可能なオブジェクト・ファイルに存在するセクション（CA850 が内部的に使用したレジスタの本数に関する情報が格納されている）。

(4) シンボル・テーブルの「Bind」

値	-v 指定時の表示	意味
0	Local	外部参照の解決において用いられることのないシンボル。
1	Global	外部参照の解決において用いられるシンボル。

(5) シンボル・テーブル「Type」

値	-v 指定時の表示	意味
1	Object	通常のオブジェクト（ラベル）。
2	Func	関数名。
3	Section	セクション。
4	File	通常のファイル名。
13	Devfile	デバイス・ファイル名。

(6) シンボル・テーブルの「Shndx」

値	-v 指定時の表示	意味
0x0	Undef	未定義なシンボル。
0xFF00	GpCommon	グローバル・ポインタ（gp）と 16 ビットのディスプレースメントを用いて参照される未定義外部シンボル。
0xFFF1	Abs	定数を示すシンボル。
0xFFF2	Common	グローバル・ポインタ（gp）と 32 ビットのディスプレースメントを用いて参照される未定義外部シンボル。

なお，オブジェクト・ファイルの形式については「付録 A オブジェクト・ファイルの形式」も参考にしてください。

第 11 章　ディスアセンブラ

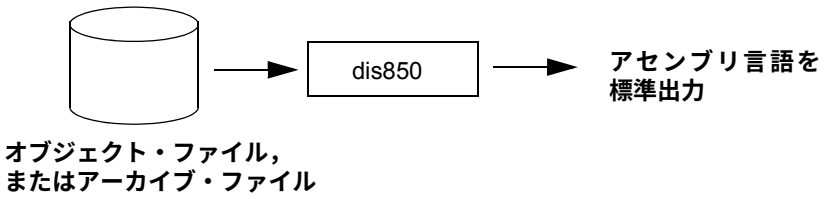
この章では，ディスアセンブラ（dis850）の概要，操作方法，出力例について説明します。

11.1　ディスアセンブラとは

ディスアセンブラとは，コンパイル／アセンブル後のオブジェクト・ファイルや，ar850 で作成されたアーカイブ・ファイルなどから，プログラム・コードをアセンブリ言語に変換し，出力するユーティリティです。オブジェクトがどのようなコードになっているかを検証したい場合などに使用します。

C コンパイラ・パッケージに入っている “ dis850 ” がディスアセンブラです。

図 11 － 1　dis850 における動作の流れ



11.2 操作方法

この節では，dis850 コマンドの操作方法について説明します。

11.2.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
dis850 [ オプション ]... ファイル名 [ ファイル名 ]...
       [ ]    ：[ ] 内は省略できます。
       ...    ：直前の [ ] 内のパターンの繰り返しができます。
```

11.2.2 PM+ による方法

ディスアセンブラ起動用のダイアログ（[オブジェクト解析ツール] ダイアログ）は，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [オブジェクト解析ツールの起動] を選択したのち，[逆アセンブラ] タブを選択

ディスアセンブラの起動は，プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

11.3 オプションの種類と機能

どのオプションも指定されなかった場合，-o オプションが指定されたものとみなします。

-A

オプション -aoptr を指定したものとみなします。

-F *devpath*

デバイス・ファイルをフォルダ *devpath* で検索します。このオプションを省略した場合，標準フォルダで検索します。

-V

dis850 のバージョン情報を標準出力に出力し，終了します。

-a

アドレスを表示します。

-c

コード（アセンブラ命令，データ）を表示します。

-e *address*

エンド・アドレスを指定します。*address* は 10 進数，または 0x を先頭に付けた 16 進数で指定します。このオプションを省略した場合，0xffffffff を指定したものとみなします。

-l *size*

表示するサイズを指定します。*size* は 10 進数，または 0x を先頭に付けた 16 進数で指定します。このオプションを省略した場合，0xffffffff を指定したものとみなします。

-m

アセンブリ言語ソースの形式で表示します。このオプションを省略した場合，シンボル・オフセットなどとともに表示します。

-o

シンボルからのオフセットを表示します。このオプションを省略した場合，-a オプション，または -m オプションが指定されていなければ表示します。

-p

プロセッサの命令フォーマットに従って並べられたコードを表示します。-c オプションが指定された場合は -c を優先します。

-r

レジスタの r0，r2，r3，r4，r5，r30，r31 を，zero，hp，sp，gp，tp，ep，lp として表示します。このオプションを省略した場合，レジスタはすべて r*num* の形で表示します。*num* は 0 から 31 の数値です。

-s *address*

スタート・アドレスを指定します。*address* は 10 進数，または 0x を先頭に付けた 16 進数で指定します。*address* の数値が 0xfffffffe より大きい場合は無視します。このオプションを省略した場合，0x0 を指定したものとみなします。

-t

表示内容を示すタイトルを表示します。

-v

コメントなどを表示します。

@*cfile*

cfile をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

11.4 PM+ での設定

この節では，対象プロジェクトのファイルに対して，dis850 のコマンド・オプションを設定するダイアログについて説明します。

11.4.1 [オブジェクト解析ツール] ダイアログ

[オブジェクト解析ツール] ダイアログの上部には，次の 3 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

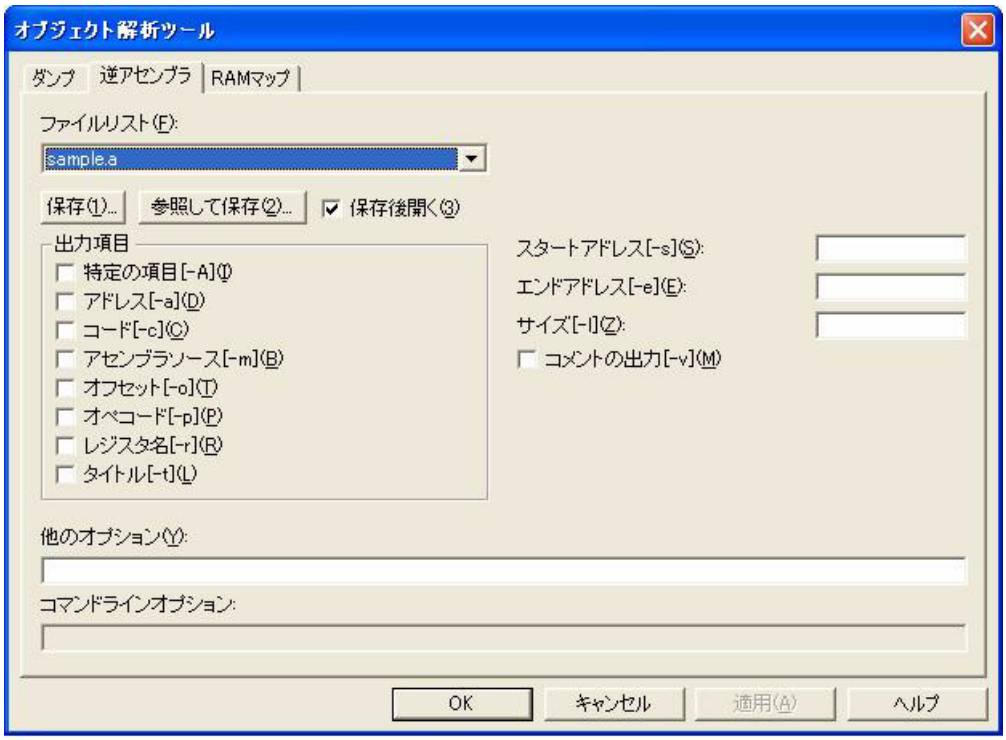
表 11 － 1 [オブジェクト解析ツール] ダイアログ（dis850）

タブ	内容
[ダンプ]	ダンプ・コマンドに関するオプションを設定
[逆アセンブラ]	ディスアセンブラに関するオプションを設定
[RAM マップ]	RAM マップに関するオプションを設定

[逆アセンブラ]

ディスアセンブラ起動用ダイアログは，プロジェクト・マネージャでプロジェクトを設定後，[ツール] メニュー→ [オブジェクト解析ツールの起動] でオープンするダイアログの [逆アセンブラ] タブを選択します。

図 11 － 2 [オブジェクト解析ツール] ダイアログ：[逆アセンブラ] タブ



(1) ファイルリスト

プロジェクトでビルド対象となるオブジェクト・ファイルをドロップダウン・リストに表示します（リストにはパスは表示されません）。

リストとして次のファイルが対象となります。

- ROM 化プロセッサの出力ファイル（実行可能オブジェクト・ファイルをビルドするプロジェクトで ROM 化を指定している場合のみ）
- リンカの出力ファイル（実行可能オブジェクト・ファイルをビルドするプロジェクトのみ）
- アーカイバの出力ファイル（ライブラリを出力するプロジェクトのみ）
- コンパイラの出力ファイル
- アセンブラの出力ファイル

[保存] ボタンにより，[名前を付けて保存] ダイアログを開き，“ ファイルリスト ”で指定したオブジェクト・ファイルの逆アセンブル結果を保存します。デフォルトでは，“ ファイルリスト ”で指定したオブジェクト・ファイルが存在するフォルダ内に，“ オブジェクト・ファイル名 .dis ” をファイル名として保存します。

なお，[参照して保存] ボタンにより，オブジェクト・ファイルを指定して逆アセンブル結果を保存することができます。この場合，[ファイルを開く] ダイアログが開き，オブジェクト・ファイルを指定します。デフォ

ルトでは，前回の [ファイルを開く] ダイアログで指定したフォルダ，またはプロジェクト・フォルダが表示されます。

[ファイルを開く] ダイアログでオブジェクト・ファイルを指定すると，指定したオブジェクト・ファイルに対する逆アセンブル結果を [保存] ボタンの選択と同様の形式で保存します。

(2) 保存後開く

[保存] ／ [参照して保存] ボタンにより逆アセンブル結果を保存したのち，PM+ 上で設定したエディタで保存したファイルを開くがどうかを指定します。デフォルトでチェックされています。

(3) 出力項目

逆アセンブル結果として出力する項目をチェック・ボックスにより選択します。

(a) 特定の項目

-A オプションで指定する，特定の項目を出力します。

チェックすることにより，-A オプションを指定した場合と同等となります。

(b) アドレス

-a オプションで指定する，アドレスを出力します。

チェックすることにより，-a オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(c) コード

-c オプションで指定する，コードを出力します。

チェックすることにより，-c オプションを指定した場合と同等となります。

(d) アセンブラソース

-m オプションで指定する，アセンブラソースを出力します。

チェックすることにより，-m オプションを指定した場合と同等となります。

(e) オフセット

-o オプションで指定する，オフセットを出力します。

チェックすることにより，-o オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(f) オペコード

-p オプションで指定する，オペコードを出力します。

チェックすることにより，-p オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(g) レジスタ名

-r オプションで指定する，レジスタ名を出力します。

チェックすることにより，-r オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(h) タイトル

-t オプションで指定する，ラインナンバ情報を出力します。

チェックすることにより，-t オプションを指定した場合と同等となります。

ただし，このチェック・ボックスは，-A オプションが指定されている場合は無効となります。

(4) スタートアドレス

-s オプションで指定する，スタート・アドレスを設定します。

エディット・ボックスにおいて，10 進／ 16 進で指定します。

指定することにより，-s オプションを指定した場合と同等となります。

(5) エンドアドレス

-e オプションで指定する，エンド・アドレスを設定します。

エディット・ボックスにおいて，10 進／ 16 進で指定します。

指定することにより，-e オプションを指定した場合と同等となります。

(6) サイズ

-l オプションで指定する，出力範囲のサイズを設定します。

エディット・ボックスにおいて，10 進／ 16 進で指定します。

指定することにより，-l オプションを指定した場合と同等となります。

(7) コメントの出力

-v オプションで指定する，コメントを出力するかどうかを指定します。

チェックすることにより，-v オプションを指定した場合と同等となります。

(8) 他のオプション

前記の “ dis850 コマンドのオプション ” では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

なお，dis850 コマンドに関するオプションは [オブジェクト解析ツール] ダイアログですべて指定できるため，他のオプションを使用する必要はありません。

(9) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

11.5 注意事項

(1) オブジェクト・ファイルに同じアドレスのラベルが存在する場合，シンボル・テーブル内で後ろに出現した方が優先されます。

(2) プログラムが 0 番地から始まっていて，かつ 0 番地を示すシンボルが存在しないオブジェクトのため，出力時に 0 番地のシンボル出力が必要とされるような場合，“ __dummy ” を 0 番地のシンボルとして出力することがあります。

11.6 出力形式

dis850 の出力の例を次に示します。

```
> dis850 -A a.out

     Address     Offset      Opecode
                                _main:
   0x00000000 : 0x00000000 : 45D5          br     _main + 0x8a
   0x00000002 : 0x00000002 : D800          mov    zero, r27
   0x00000004 : 0x00000004 : E6230000      movea  0, sp, r28
   0x00000008 : 0x00000008 : 301C          mov    r28, r6
   0x0000000A : 0x0000000A : FF800176      jarl   _getToken[pc], lp
   0x0000000E : 0x0000000E : 580A          mov    r10, r11
   0x00000010 : 0x00000010 : 5A7F          cmp    -0x1, r11
   0x00000012 : 0x00000012 : 1D92          bz     _main + 0x44
   0x00000014 : 0x00000014 : EE2003E8      movea  0x3e8, zero, r29
   0x00000018 : 0x00000018 : D9FD          cmp    r29, r27
   0x0000001A : 0x0000001A : 15DE          bge    _main + 0x44
   0x0000001C : 0x0000001C : 301C          mov    r28, r6
   0x0000001E : 0x0000001E : FF800000      jarl   0[pc], lp
   0x00000022 : 0x00000022 : 580A          mov    r10, r11
   0x00000024 : 0x00000024 : 501B          mov    r27, r10
   0x00000026 : 0x00000026 : 52C2          shl    0x2, r10
   0x00000028 : 0x00000028 : 66230020      movea  0x20, sp, r12
   0x0000002C : 0x0000002C : 61CA          add    r10, r12
   0x0000002E : 0x0000002E : 5F6C0001      st.w   r11, 0[r12]
   0x00000032 : 0x00000032 : DA41          add    0x1, r27
   0x00000034 : 0x00000034 : 301C          mov    r28, r6
   0x00000036 : 0x00000036 : FF80014A      jarl   _getToken[pc], lp
   0x0000003A : 0x0000003A : 580A          mov    r10, r11
   0x0000003C : 0x0000003C : 5A7F          cmp    -0x1, r11
   0x0000003E : 0x0000003E : 05B2          bz     _main + 0x44
   ...
```

a.out に含まれている情報のうち，アドレス，オフセット，命令フォーマット形式に従ったコード，タイトルをアセンブリ言語の命令とともに表示し，レジスタは別名で表示します。

第 12 章　クロス・リファレンス・ツール

この章では，クロス・リファレンス・ツール（cxref）の概要，操作方法，出力ファイルの形式について説明します。

12.1　クロス・リファレンス・ツールとは

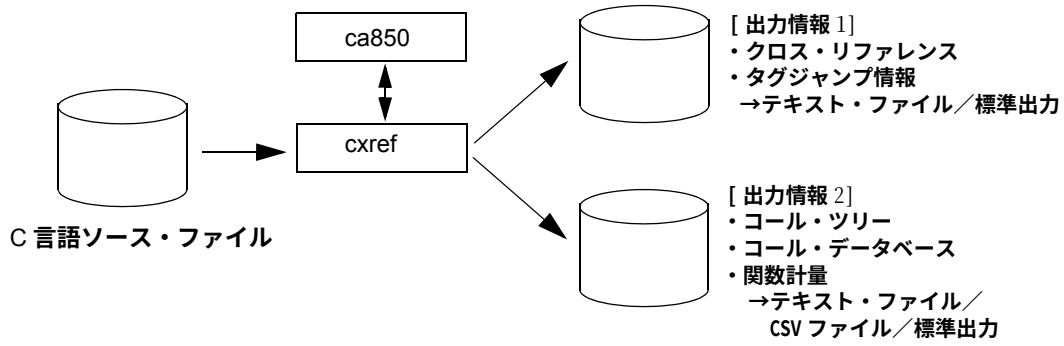
クロス・リファレンス・ツール“ cxref ”は，C 言語ソース・ファイルを基に，識別子の参照と定義位置を検出するツールです。対象となる識別子は，関数，および変数（auto 変数以外）であり，その記憶クラスも識別します。検出結果として，クロス・リファレンス情報と，タグ・ジャンプ情報を出力します。また，関数単位の解析を行い，コール・ツリー，関数計量，コール・データベースを出力することもできます。

クロス・リファレンス・ツールの処理において，“ 参照 ”とは，式の中にその識別子が現れることをいいます。“ 定義 ”とは，宣言文の中にその識別子が現れることをいいます。式か宣言文か判断ができないものについては，クロス・リファレンスは“ 不明 ”として扱います。

クロス・リファレンス・ツールが出力するコール・ツリー，関数計量，コール・データベースは，次のような特長があります。

- ターゲット，および ca850 の最適化に依存しない
- オプションにより，標準出力が可能

図 12 － 1　cxref における動作の流れ



12.2 入出力

12.2.1 入力ファイル

cxref の入力ファイルは，C 言語ソース・ファイルです。クロス・リファレンス・ツールの起動時に，“ -cpp850 ” オプションを指定すると，指定した C 言語ソース・ファイルをプリプロセッサに通したあと，クロス・リファレンス・ツールが処理を行います。

(1) cxref は，入力する C 言語ソース・ファイルに構文エラーが含まれないことを前提に処理を行います。
C 言語ソース・ファイルは一度コンパイルを実行し，構文エラーがないことを確認してください。

(2) 文字セットは，シフト JIS とします。

(3) クロス・リファレンス・ツールは，C 言語ソース・ファイルに含まれるプリプロセス指令をエラー扱いせず，単に無視して解析を行います。したがって，次のものを含まない C 言語ソース・ファイルであれば，ca850 を通っていないファイルであっても，“ -cpp850 ” オプション指定せず，そのまま処理することができます。これは，ヘッダ・ファイルを無視したい場合，偽条件ブロック内も解析の対象としたい場合，およびマクロ名をクロス・リファレンスの対象としたい場合に有効です。

- “ { } ” のバランスを乱す条件ブロック
- 制御構造のマクロ化
- 宣言文のマクロ化

(4) 入力ファイルには行番号情報とコメント情報を含めることができます。

12.2.2 出力情報

cxref が出力する情報は，次のとおりです。

(1) **クロス・リファレンス**
ファイルごとに，そのファイル内で使用されている変数，および関数のクロス・リファレンス情報を出力します。

(2) **タグ情報**
変数，および関数について，その定義ファイル名と行番号の情報（タグ・ジャンプ情報）を出力します。

(3) **コール・ツリー**
ある関数が，どの関数を呼び出しているかをツリー状に出力します。

(4) **関数計量**
関数の “ ライン数 ”“ 呼び出される頻度 ” の情報を出力します。

(5) **コール・データベース**
ある関数が，どの関数を何回呼び出しているかを出力します。

これらの情報についての詳細は「12.6 出力形式」を参照してください。

12.3 操作方法

この節では，cxref 操作方法について説明します。

12.3.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
cxref [ オプション ]... [ ファイル名 ]...
       [ ]    ：[ ] 内は省略できます。
       ...    ：直前の [ ] 内のパターンの繰り返しができます。
```

12.3.2 PM+ による方法

クロス・リファレンス設定用ダイアログ（[静的性能解析ツール] ダイアログ）は，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [静的性能解析ツールの設定] を選択したのち，[クロスリファレンス] タブを選択

クロス・リファレンス・ツールの起動は，プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

12.4 オプションの種類と機能

この節では，cxref のオプションを示します。

それぞれの出力情報別に記載してあります。

12.4.1 共通オプション

次に，cxref の共通オプションを示します。

-v

cxref の版番号を出力し，終了します。

-all

すべての情報を，テキスト形式，および CSV 形式の各ファイルに出力します。“ -x -t -c -cc -m -mc -b -bc ” を指定したのと同じです。

-cpp850

C 言語ソース・ファイルを ca850（プリプロセッサ）に通してから処理します。

なお，このオプション以降のオプションは，すべて ca850 のオプションとして渡されます。したがって，このオプションは，クロス・リファレンス・ツール・オプションとしては，最後に指定する必要があります。

また，行番号をより正確に出力するために，プリプロセッサで，ソース・プログラムのコメントを含める “ -C ” オプションを指定することを推奨します。

例

```
> cxref -t -cpp850 -cpu 3201 -DDEBUG -I..\myinc main.c sub.c
```

上記の操作は，次の操作と同じです。

```
> ca850 -cpu 3201 -E -DDEBUG -I..\myinc main.c > main.i
> ca850 -cpu 3201 -E -DDEBUG -I..\myinc sub.c > sub.i
> cxref -t main.i sub.i
```

-*dident*

型名として扱う識別子を指定します。

-d=*file*

型名として扱う識別子を記述したファイル名 *file* を指定します。

-file=*file*

次の情報を記述したファイル名 *file* を指定します。

(1) 実行結果に表示しないファイル名
(2) 実行結果に表示しない識別子名
(3) 型名として扱う識別子名

-file=*file* と -ni を同時に指定した場合，前に指定した -file=*file* の *file* における “ NoIncludeFile ” の内容は無効となります。

- -ni ／ -i ／ -d ／ -file オプションに指定するファイルの形式

  -ni ／ -i ／ -d でそれぞれのオプションで該当するセクション情報を読み出し，-file オプションによりすべてのセクション情報を読み出します。

  次の 3 つのセクションを記述できます。

  - NoIncludeFile セクション
  - DefinitionType セクション

- IgnoreIdent セクション

なお，行頭が “ // ” で始まる行は，コメントとして扱います。

(1) NoIncludeFile セクション

解析結果として表示しない情報をファイル単位で指定します。主にインクルード・ファイルを記述します。ここに記述したファイル名は -ni オプションに続いて指定する場合と同じ効果があります。

ファイル名は 1 行に 1 つ記述します。ワイルド・カードが使用可能です。

例

```
[NoIncludeFile]
// *.h のファイルすべて
*.h
// 共通の定義ファイル
common.def
```

(2) DefinitionType セクション

型名として扱う識別子を指定します。ここに記述したファイル名は -d オプションに続いて指定する場合と同じ効果があります。識別子は 1 行に 1 つ記述します。

例

```
[DefinitionType]
// 1 バイトの型
BYTE
UBYTE
// 2 バイトの型
WORD
UWORD
```

(3) IgnoreIdent セクション

解析結果として表示しない情報を識別子単位で指定します。ここに記述したファイル名は -i オプションに続いて指定する場合と同じ効果があります。識別子は 1 行に 1 つ記述します。

例

```
[IgnoreIdent]
// 各処理で一時的に使用する共通エリア
tmp
buf
work
```

-h

-help

オプションの説明を出力し，終了します。

-i*ident*

実行結果に表示しない識別子を指定します。

-ni

インクルード・ファイルの情報を表示しません。

-ni*file*

実行結果に表示しないファイル名 *file* を指定します。*file* には，次のワイルド・カードを使用することができます。

?	任意の 1 文字
*	0 文字以上の任意の文字列

例

r	'r' を含むファイル名
e??	'e' を含み，その後に 2 文字以上の文字があるファイル名
w?*.h	'w' で始まる 2 文字以上で末尾が " .h " のファイル名

-ni=*file*

実行結果に表示しないファイル名を記述したファイル名 *file* を指定します。

-o *path*

出力ファイルのパス *path* を指定します。このオプションを省略した場合，カレント・パスに出力されます。

@*cfile*

cfile をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

12.4.2 クロス・リファレンス

次に，cxref のクロス・リファレンス用オプションを示します。

-x[=*file*]

クロス・リファレンスをテキスト形式で，指定したファイルに出力します。“ *=file* ” を省略した場合，ファイル名は cxref になります。

-xstd

クロス・リファレンスを標準出力に出力します（デフォルト）。

12.4.3 タグ情報

次に，cxref のタグ情報用オプションを示します。

-t[=*file*]

タグ情報をテキスト形式で，指定したファイル *file* に出力します。“ *=file* ” を省略した場合，ファイル名は ctags になります。

-tstd

タグ情報を標準出力に出力します。

12.4.4 コール・ツリー

次に，cxref のコール・ツリー用オプションを示します。

-c[=*file*]

コール・ツリーをテキスト形式で，指定したファイル *file* に出力します。“ =*file* ” を省略した場合，ファイル名は ccalltre.lst になります。

-cc[=*file*]

コール・ツリーを CSV 形式で，指定したファイル *file* に出力します。“ =*file* ” を省略した場合，ファイル名は ccalltre.csv になります。

-call[=*file*]

コール・ツリーをテキスト形式と CSV 形式で，指定したファイルに出力します。ファイル名は *file*.lst，および *file*.csv になります。*file* に拡張子を付けて指定した場合，その拡張子は無視されます。“ =*file* ” を省略した場合，ファイル名は ccalltre.lst，および ccalltre.csv になります。

-ce*num*

出力の省略方法を指定します。*num* には，次のいずれかが指定可能です。

1	すべて出力する。
2	同一レベルのコール・ツリーの場合，省略する。
3	一度出力したら省略する。

このオプションを省略した場合，-ce3 を指定したものとみなします。

-cf*string*

コール・ツリーを出力する関数名を *string* に指定します。

-cf=*file*

コール・ツリーを出力する関数名を記述した，テキスト・ファイル *file* を指定します。

-cl*num*

出力レベルを指定します。*num* には，1 ～ 255 が指定可能です。このオプションを省略した場合，255 を指定したものとみなします。

-cp

引数，戻り値を含めて出力します。

-cr

参照情報を含めて出力します。

-cs

ソース・ファイル名，記述開始行を含めて出力します。

-cstd

テキスト形式のコール・ツリーを標準出力に出力します。

-ct

先頭のツリーのみ出力します。

12.4.5 関数計量

次に，cxref の関数計量用オプションを示します。

-m[=*file*]

関数計量をテキスト形式で，指定したファイル *file* に出力します。“ *=file* ” を省略した場合，ファイル名は cmeasure.lst になります。

-mc[=*file*]

関数計量を CSV 形式で，指定したファイル *file* に出力します。“ *=file* ” を省略した場合，ファイル名は cmeasure.csv になります。

-mall[=*file*]

関数計量をテキスト形式と CSV 形式で，指定したファイルに出力します。ファイル名は *file*.lst，および *file*.csv になります。*file* に拡張子を付けて指定した場合，その拡張子は無視されます。“ *=file* ” を省略した場合，ファイル名は cmeasure.lst，および cmeasure.csv になります。

-ms[+|-] *num*

出力順序を指定します。*num* には，次のいずれかが指定可能です。

1	関数名のアルファベット順にソートして出力する。
2	ファイル名，関数名のアルファベット順にソートして出力する。
3	ソートしない。

‘ + ’ を指定した場合，昇順に出力します。‘ - ’ を指定した場合，降順に出力します。デフォルトは，降順です。このオプションを省略した場合，ソートされず，関数の出現順に出力されます。

-mstd

テキスト形式の関数計量を標準出力に出力します。

12.4.6 コール・データベース

次に，cxref のコール・データベース用オプションを示します。

-b[=*file*]

コール・データベースをテキスト形式で，指定したファイル *file* に出力します。“ *=file* ” を省略した場合，ファイル名は cprofile.dat になります。

-bc[=*file*]

コール・データベースを CSV 形式で，指定したファイル *file* に出力します。“ *=file* ” を省略した場合，ファイル名は cprofile.csv になります。

-ball[=*file*]

コール・データベースをテキスト形式と CSV 形式で，指定したファイルに出力します。ファイル名は *file*.lst，および *file*.csv になります。*file* に拡張子を付けて指定した場合，その拡張子は無視されます。“ *=file* ” を省略した場合，ファイル名は cprofile.dat，および cprofile.csv になります。

-bstd

テキスト形式のコール・データベースを標準出力に出力します。

12.5 PM+ での設定

この節では，対象プロジェクトのファイルに対して，cxref のコマンド・オプションを設定するダイアログについて説明します。

12.5.1 [静的性能解析ツール] ダイアログ

[静的性能解析ツール] ダイアログの上部には，次の 2 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

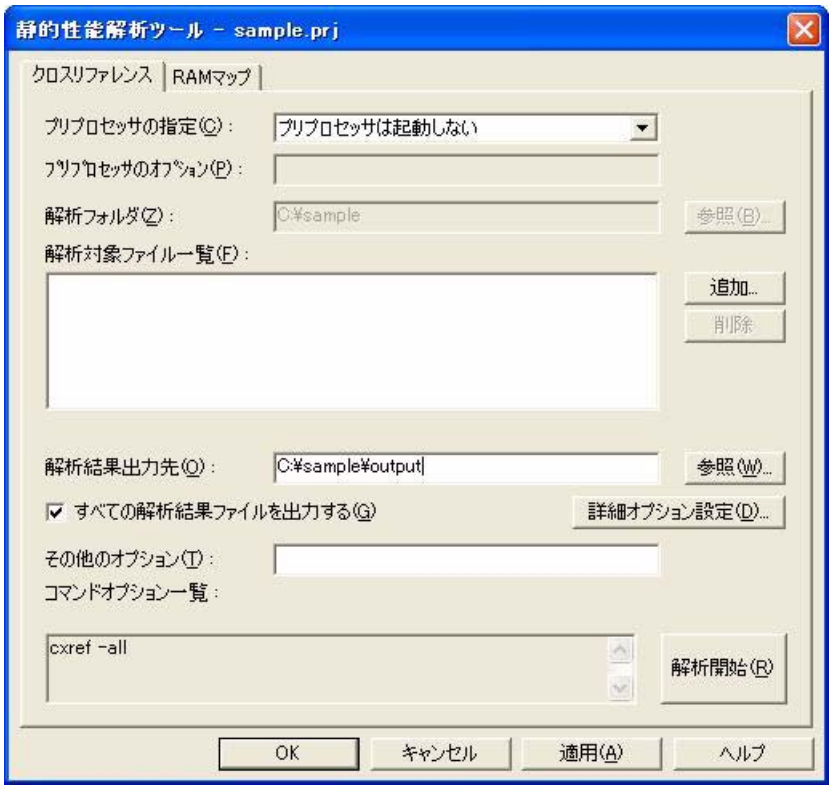
表 12 － 1 [静的性能解析ツール] ダイアログ（cxref）

タブ	内容
[クロスリファレンス]	クロス・リファレンス関するオプションを設定
[RAM マップ]	RAM マップ関するオプションを設定

[クロスリファレンス]

クロス・リファレンス設定用ダイアログは，PM+ でプロジェクトを設定後，[ツール] メニュー→ [静的性能解析ツールの設定] でオープンするダイアログの [クロスリファレンス] タブを選択します。

図 12 － 2 [静的性能解析ツール] ダイアログ：[クロスリファレンス] タブ



(1) プリプロセッサの指定

エディット・ボックスによりプリプロッセッサを選択します。デフォルトは，次のようになります。

- オプションを保存している場合は，保存しているプリプロセッサの指定内容を表示します。
- オプションを保存していないが，プロジェクト・ファイル内のコンパイラ・オプションで指定した情報がある場合は，「プロプロセッサを起動する」になります。
- オプションを保存せず，プロジェクト・ファイル内のコンパイラ・オプションで指定した情報もない場合は，「プロプロセッサは起動しない」になります。

(2) プリプロセッサのオプション

プリプロセッサのオプションを指定してください。

なお，行番号をより正確に出力するために，プリプロセッサで，ソース・プログラムのコメントを含める “ -C ” オプションを指定することを推奨します。

(3) 解析フォルダ

エディット・ボックスにより解析フォルダを指定します。デフォルトは，次のようになります。

- オプションを保存している場合は，保存している解析フォルダになります。
- オプションを保存していない場合は，プロジェクト・ファイルが置かれているフォルダになります。

(4) 解析対象ファイル一覧

リスト・ボックスに解析対象ファイルの一覧が表示されます。デフォルトは，次のようになります。

- オプションを保存している場合は，保存している解析対象ファイルになります。
- オプションを保存していない場合は，プロジェクト・ファイルに登録されているソース・ファイルに対応するプロジェクト・ファイルに 指定されている C 言語ソース・ファイルになります。

[[追加] ボタンにより [解析対象ファイルの設定] ダイアログが開き，ファイルを追加指定します。指定したファイルは一覧に表示されます。“ 解析フォルダ ” が空欄の場合，指定したファイルのパスが “ 解析フォルダ ” に表示されます。

[解析対象ファイルの設定] ダイアログでは，次の順序でデフォルトの位置が決定されます。

(a) 解析フォルダが指定されている場合，解析フォルダ
(b) [クロスリファレンス] タブの " 解析結果出力先 "
(c) [RAM マップ] タブの “ 解析対象ファイル ” のパス
(d) インストール・フォルダ

[削除] ボタンにより，一覧で選択したファイルが削除されます。[削除] ボタンは，一覧でファイルを指定した場合のみ選択できます。

(5) 解析結果出力先

エディット・ボックスにより解析結果出力先（フォルダ）を指定します。デフォルトは，次のようになります。

- オプションを保存している場合は，保存している解析結果出力先になります。
- オプションを保存していない場合は，プロジェクト・ファイルのおかれているフォルダになります。

指定されたフォルダが存在しないとき，[OK] ボタン，または [適用] ボタンにより [フォルダ作成] ダイアログが開き，[OK] ボタンを選択するとフォルダが作成されます。

[参照] ボタンにより [フォルダの参照] ダイアログが開き，解析結果出力先を選択します。選択したフォルダは “ 解析結果出力先 ” に表示されます。

なお，[フォルダの参照] ダイアログでは，次の順序でデフォルトの位置が決定されます。

(a) 解析結果出力先フォルダが指定されている場合，解析結果出力先フォルダ
(b) 解析フォルダが指定されている場合，解析フォルダ
(c) [RAM マップ] タブの “ 解析対象ファイル ” のパス
(d) インストール・フォルダ

(6) すべての解析結果ファイルを出力する

チェックされている場合は，オプション内容をデフォルトの内容で設定します。デフォルトでは，チェックされています。

(7) 詳細オプション設定

[クロスリファレンスのオプション設定] ダイアログが開き，各機能のオプション内容を設定できます。

(8) その他のオプション

エディット・ボックスにより直接オプションを記述できます。デフォルトでは，空欄です。

(9) コマンド・オプション一覧

オプションの一覧が表示されます。デフォルトは，“ cxref -all ” です。

[解析開始] ボタンにより，設定されたオプション内容で解析対象ファイルが解析されます。インストール・フォルダにある解析コマンドが起動されます。ただし，解析対象ファイルの指定がない場合，解析ファイルの指定がないことを知らせるメッセージ・ボックスが開きます。

解析の結果によって，正常終了，または異常終了を知らせるメッセージ・ボックスが開きます（解析コマンドの出力メッセージは，解析結果出力先のフォルダ，または解析フォルダのファイル cxref.log に出力されます）。

12.5.2 [クロスリファレンスのオプション設定] ダイアログ

[クロスリファレンスのオプション設定] ダイアログの上部には，次の 6 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

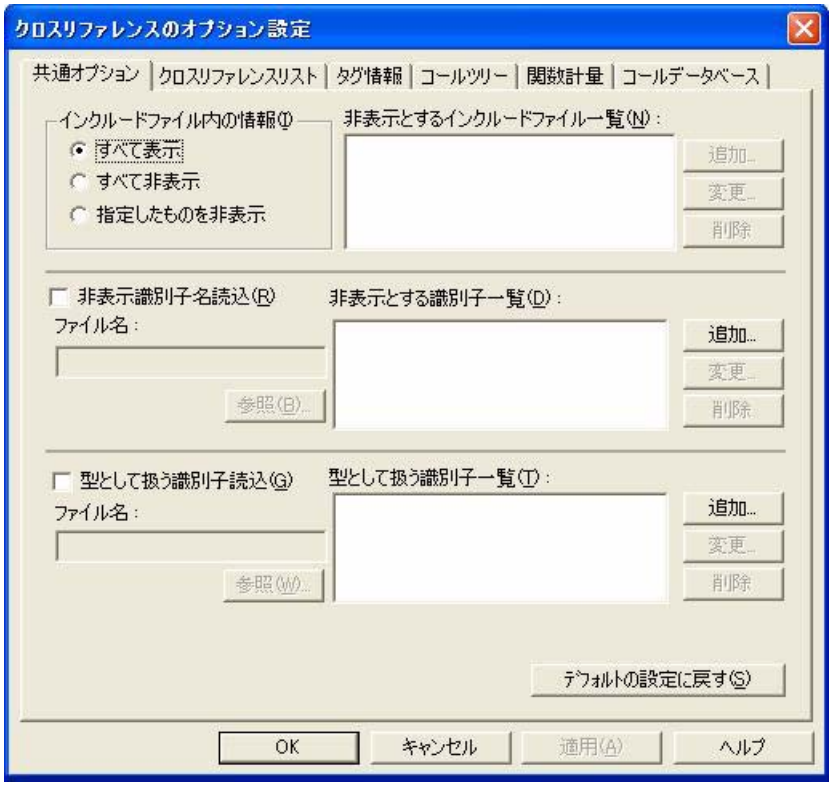
表 12 － 2 [クロスリファレンスのオプション設定] ダイアログ

タブ	内容
[共通オプション]	クロス・リファレンス全体に関するオプション設定
[クロスリファレンスリスト]	ファイルごとに，そのファイルで使用されている変数，および関数のクロス・リファレンス情報の解析に関するオプション設定
[タグ情報]	変数，および関数について，その定義ファイル名と行番号の情報（タグ・ジャンプ情報）の解析に関するオプション設定
[コールツリー]	ある関数がどの関数を呼び出しているかの解析に関するオプション設定
[関数計量]	出力形式の指定，ソート方法，ソート方向に関するオプション設定
[コールデータベース]	ある関数が，どの関数を何回呼び出しているかの解析に関するオプション設定

[共通オプション]

クロス・リファレンス全体に関するオプション設定を行います。

図 12 － 3 [クロスリファレンスのオプション設定] ダイアログ：[共通オプション] タブ



(1) インクルードファイル内の情報

インクルード・ファイル情報の実行結果への出力方法を指定します。

すべて表示	インクルード・ファイル内の情報をすべて実行結果に表示します（デフォルト）。
すべて非表示	インクルード・ファイル内の情報をすべて表示しません。
指定したものを非表示	“ 非表示とするインクルード・ファイル一覧 ” と [追加] ボタンが選択できるようになります。

(2) 非表示とするインクルードファイル一覧

非表示とするインクルード・ファイル名が表示されます。38 文字以上のファイル名は先頭から 18 文字＋“ ... ”＋最後から 18 文字が表示されます。“ インクルードファイル内の情報 ” で “ 指定したものを非表示 ” を指定した場合に選択できます。

[追加] ボタンにより [解析対象ファイルの設定] ダイアログが開きます。そこでインクルード・ファイルを選択すると一覧に追加されます。

[解析対象ファイルの設定] ダイアログでは，次の順序でデフォルトの位置が決定されます。

(1) [クロスリファレンス] タブの “ 解析フォルダ ”

(2) [RAM マップ] タブの “ 解析結果出力先 ” のパス

(3) インストール・フォルダ

一覧でインクルード・ファイルを指定してのち，[変更] ボタンを選択すると [解析対象ファイルの設定ダイアログ] が開き，ダイアログで選択したインクルード・ファイルに変更されます。

[削除] ボタンにより，一覧から選択したインクルード・ファイルが削除されます。

ただし，[変更] ボタン，および [削除] ボタンは，一覧でインクルード・ファイルを指定した場合のみ選択できます。

(3) 非表示識別子名読込

情報を出力しない識別子名が表示されます。このチェック・ボックスをオンにし，ファイル名を指定することにより非表示にする識別子名を読み込みます。“ ファイル名 ” は，デフォルトでは，空欄です。ダイアログ下部の [適用] ボタンを選択すると,指定したファイル内の識別子名が“ 非表示とする識別子一覧 ”に表示されます。

[参照] ボタンを選択すると，[ファイルの参照] ダイアログで既存のファイル名を参照することができます。

(4) 非表示とする識別子一覧

非表示とする識別子名が表示されます。38 文字以上のファイル名は先頭から 18 文字＋ “ ... ” ＋最後から 18 文字が表示されます。

[追加] ボタンにより [解析対象ファイルの設定] ダイアログが開き，識別子を選択すると一覧に追加されます。一覧で識別子を指定して [変更] ボタンを選択すると [解析対象ファイルの設定] ダイアログが開き，ダイアログで選択した識別子に変更されます。

[削除] ボタンにより，一覧から選択した識別子が削除されます。

ただし，[変更] ボタン，および [削除] ボタンは，一覧で識別子を指定した場合のみ選択できます。

(5) 型として扱う識別子読込

型として扱う識別子名が表示されます。このチェック・ボックスをオンにし，ファイル名を指定することにより型として扱う識別子名を読み込みます。“ ファイル名 ” は，デフォルトでは，空欄です。ダイアログ下部の [適用] ボタンを選択すると，指定したファイル内の識別子名が“ 型として扱う識別子名一覧 ”に表示されます。

[参照] ボタンを選択すると，[ファイルの参照] ダイアログで既存のファイル名を参照することができます。

(6) 型として扱う識別子一覧

型として扱う識別子名が表示されます。38 文字以上のファイル名は先頭から 18 文字＋“...”＋最後から 18 文字が表示されます。

[追加] ボタンにより [解析対象ファイルの設定] ダイアログが開き，識別子を選択すると一覧に追加されます。一覧で識別子を指定して [変更] ボタンを選択すると [解析対象ファイルの設定] ダイアログが開き，ダイアログで選択した識別子に変更されます。

[削除] ボタンにより，一覧から選択した識別子が削除されます。

ただし，[変更] ボタン，および [削除] ボタンは，一覧で識別子を指定した場合のみ選択できます。

[ボタン]

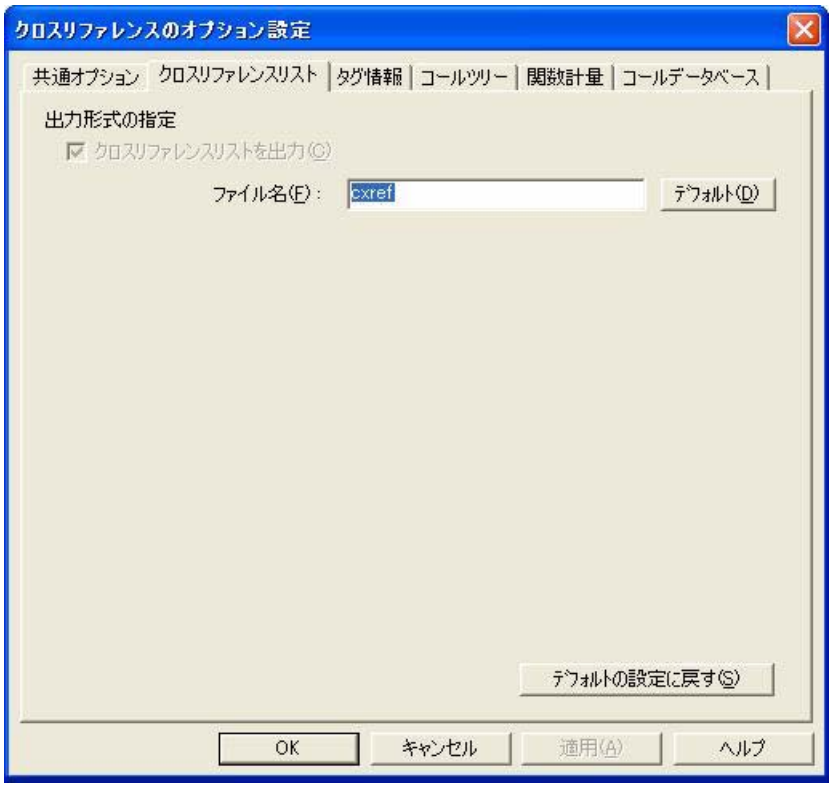
(1) [デフォルトの設定に戻す] ボタン

現在のダイアログの設定をデフォルトの状態に戻します。

[クロスリファレンスリスト]

ファイルごとに，そのファイルで使用されている変数，および関数のクロス・リファレンス情報の解析に関するオプション設定を行います。

図 12 － 4 [クロスリファレンスのオプション設定] ダイアログ：[クロスリファレンスリスト] タブ



(1) 出力形式の指定

“ クロスリファレンスリストを出力 ”をチェックされている場合は，解析結果がテキスト形式で出力されます。デフォルトでは，チェックされていません。“ ファイル名 ” エディット・ボックスによりテキスト形式の解析結果を出力するファイルを指定します。デフォルトは cxref です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。

“ クロスリファレンスリストを出力 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

ただし，[クロスリファレンス] タブで “ すべての解析結果ファイルを出力する ” をチェックしている場合，“ クロスリファレンスリストを出力 ” はオンのままとなります。

[ボタン]

(1) [デフォルトの設定に戻す] ボタン

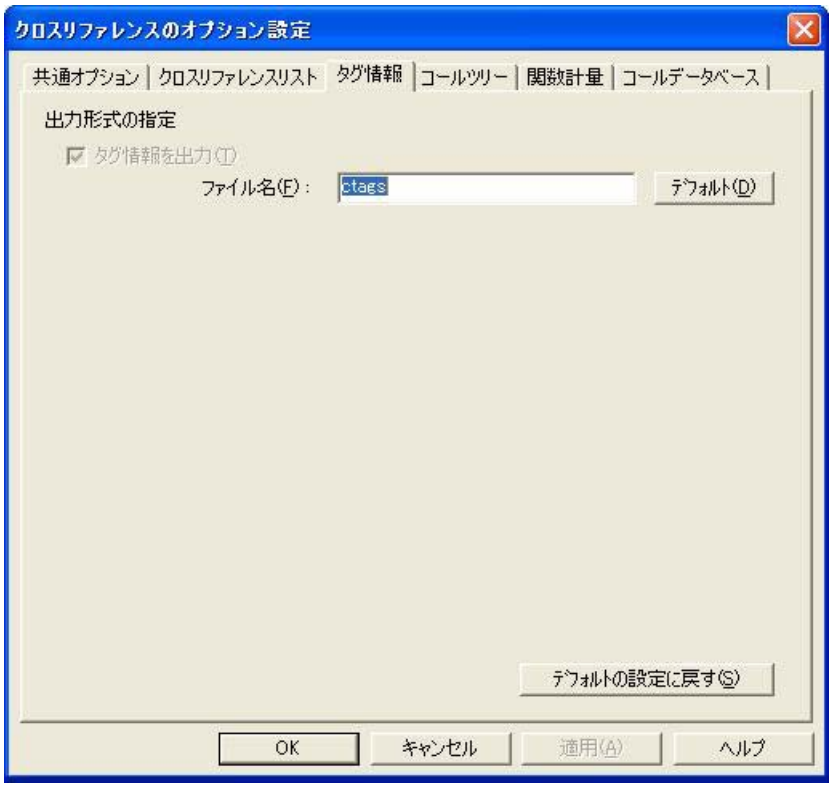
現在のダイアログの設定をデフォルトの状態に戻します。

ただし，[クロスリファレンス] タブで “ すべての解析結果ファイルを出力する ” をチェックしている場合，“ クロスリファレンスリストを出力 ” はオンのままとなります。

[タグ情報]

変数，および関数について，その定義ファイル名と行番号の情報（タグ・ジャンプ情報）の解析に関するオプション設定を行います。

図 12 － 5 [クロスリファレンスのオプション設定] ダイアログ：[タグ情報] タブ



(1) 出力形式の指定

“ タグ情報を出力 ” をチェックされている場合は，タグ情報の解析結果がテキスト形式で出力されます。デフォルトでは，チェックされていません。“ ファイル名： ” エディット・ボックスによりテキスト形式の解析結果を出力するファイルを指定します。デフォルトは ctags です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。

“ タグ情報を出力 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

ただし，[クロスリファレンス] タブで “ すべての解析結果ファイルを出力する ” をチェックしている場合，“ クロスリファレンスリストを出力 ” はオンのままとなります。

[ボタン]

(1) [デフォルトの設定に戻す] ボタン

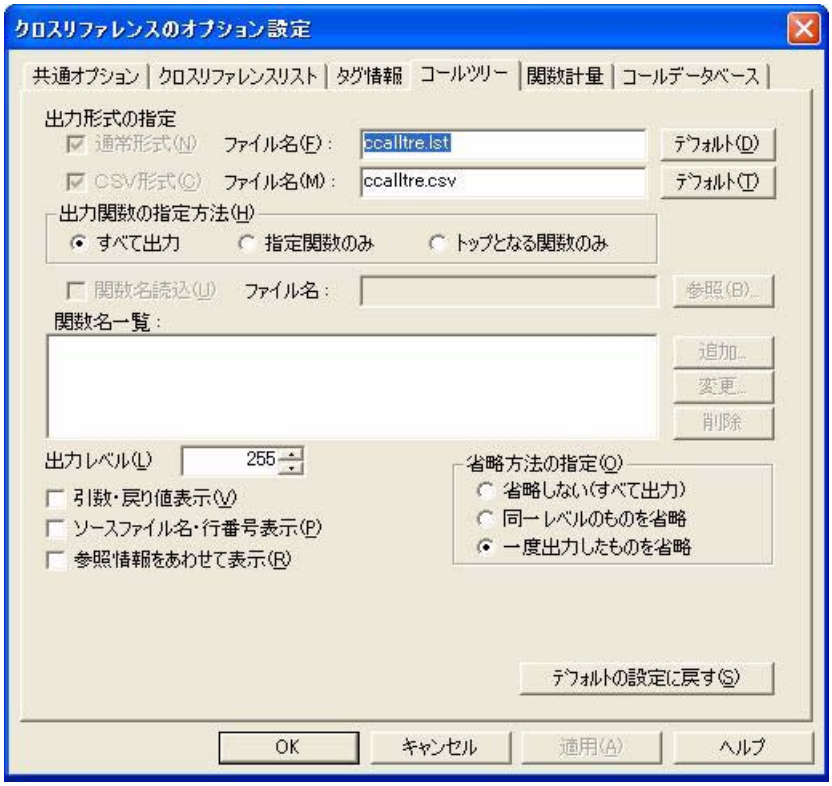
現在のダイアログの設定をデフォルトの状態に戻します。

ただし，[クロスリファレンス] タブで “ すべての解析結果ファイルを出力する ” をチェックしている場合，“ タグ情報を出力 ” はオンのままとなります。

[コールツリー]

ある関数がどの関数を呼び出しているかの解析に関するオプション設定を行います。

図 12 － 6 [クロスリファレンスのオプション設定] ダイアログ：[コールツリー] タブ



(1) 出力形式の指定

“ 通常形式 ” をチェックされている場合は，コール・ツリーの解析結果がテキスト形式で出力されます。デフォルトでは，チェックされていません。“ ファイル名 ” エディット・ボックスにより通常形式の解析結果を出力するファイルを指定します。デフォルトは ccalltre.lst です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。

“ 通常形式 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

“ CSV 形式 ” をチェックされている場合は，コール・ツリーの解析結果が CSV 形式で出力されます。デフォルトでは，チェックされていません。

“ ファイル名 ” エディット・ボックスにより CSV 形式の解析結果を出力するファイルを指定します。デフォルトは ccalltre.csv です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。

“ CSV 形式 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

(2) 出力関数の指定方法

出力関数の指定方法を選択します。

すべて出力	すべての関数が出力されます（デフォルト）。
指定関数のみ	指定された関数のみ出力されます。 選択した場合，“ 関数名読込 ”，“ 関数名一覧 ”，および “ 関数名一覧 ” の [追加] ボタンが選択できるようになります。
トップとなる関数のみ	ツリーのトップとなる関数のみ出力されます。

(3) 関数名読込

チェックされている場合は，ファイルによる関数名指定を行います。デフォルトではチェックされていません。“ ファイル名 ”エディット・ボックスにより関数を記述したファイルを指定します。デフォルトは空欄です。

[参照] ボタンにより [解析対象ファイルの設定] ダイアログが開き，関数が記述されたファイルを選択します。[解析対象ファイルの設定] ダイアログでは，次の順序でデフォルトの位置が決定されます。

(a) [クロスリファレンス] タブの “ 解析フォルダ ”

(b) [RAM マップ] タブの “ 解析結果出力先 ” のパス

(c) インストール・フォルダ

“ 出力関数の指定方法 ” で “ 指定関数のみ ” を選択した場合のみ，この設定ボックスと “ ファイル名： ” エディット・ボックス，および [参照] ボタンは選択できます。

ファイル選択後，[適用] ボタンにより，指定したファイルで指定されている関数が “ 関数名一覧 ” に表示されます。

(4) 関数名一覧

リスト・ボックスに出力関数名が表示されます。デフォルトでは空欄です。38 文字以上の関数名は先頭から 18 文字＋ “ ... ” ＋最後から 18 文字が表示されます。“ 関数名読込 ” がチェックされている場合は，この設定リスト・ボックスは選択できません。

[追加] ボタンにより関数名が追加表示され，直接入力はできません。[追加] ボタンの選択により [関数名の指定] ダイアログが開き，関数を指定します。指定した関数は一覧に表示されます。

一覧で関数を選択したのち，[変更] ボタンを選択すると [関数名の指定] ダイアログが開き，そこで選択した関数に変更されます。

[削除] ボタンにより，一覧で選択した関数が削除されます。

ただし，[変更] ボタン，および [削除] ボタンは，一覧で関数を指定した場合のみ選択できます。

(5) 出力レベル

エディット・ボックスで解析結果の出力レベルを指定します。また，スピン・ボタンによる変更もできます。数値のみ指定でき，設定可能範囲は 1 ～ 255 で，デフォルトでは 255 です。

指定値が範囲外の場合，[OK] ボタン，または [適用] ボタンが選択された際に，指定範囲外であることを知らせるメッセージ・ボックスが開きます。

(6) 引数・戻り値表示

チェックされている場合は，関数の引数，戻り値が表示されます。デフォルトではチェックされていません。

(7) ソースファイル名・行番号表示

チェックされている場合は，ソース・ファイル名，記述開始行が表示されます。デフォルトではチェックされていません。

(8) 参照情報をあわせて表示

チェックされている場合は，関数の参照情報が表示されます。デフォルトではチェックされていません。

(9) 省略方法の指定

省略方法を指定します。

省略しない	解析結果の省略を行いません。
同一レベルのものを省略	ツリー上の同一レベルにある関数を出力しません
一度出力したものを省略	一度出力された関数を出力しません（デフォルト）。

[ボタン]

(1) [デフォルトの設定に戻す] ボタン

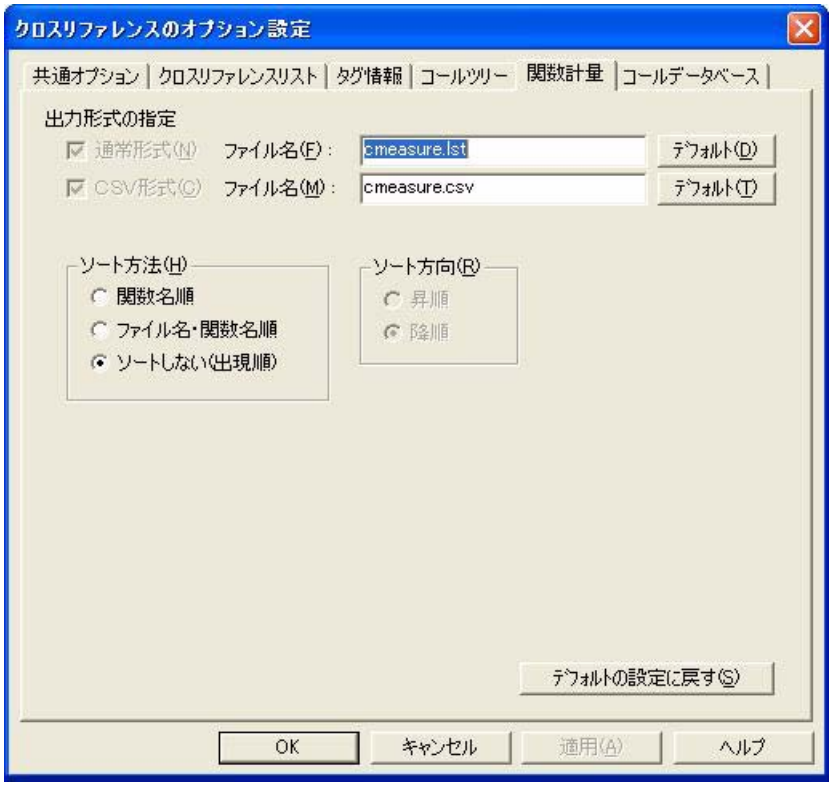
現在のダイアログの設定をデフォルトの状態に戻します。

ただし，[クロスリファレンス] タブで“ すべての解析結果ファイルを出力する ”をチェックしている場合，“ 通常形式 ” と “ CSV 形式 ” はオンのままとなります。

[関数計量]

出力形式の指定，ソート方法，ソート方向に関するオプション設定を行います。

図 12 － 7 [クロスリファレンスのオプション設定] ダイアログ：[関数計量] タブ



(1) 出力形式の指定

“ 通常形式 ” をチェックされている場合は，リーフ・リストの解析結果がテキスト形式で出力されます。デフォルトでは，チェックされていません。

“ ファイル名： ” エディット・ボックスにより通常形式の解析結果を出力するファイルを指定します。デフォルトは cmeasure.lst です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。“ 通常形式 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

“ CSV 形式 ” をチェックされている場合は，リーフ・リストの解析結果が CSV 形式で出力されます。デフォルトでは，チェックされていません。

“ ファイル名： ” エディット・ボックスにより CSV 形式の解析結果を出力するファイルを指定します。デフォルトは cmeasure.csv です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。

“ CSV 形式 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

ただし，[クロスリファレンス] タブで “ すべての解析結果ファイルを出力する ” をチェックしている場合，“ 通常形式 ” と “ CSV 形式 ” はオンのままとなります。

(2) ソート方法

解析結果のソート方法を指定します。デフォルトでは，ソートしません（出現順)。“ ソートしない (出現順) ” を選択した場合，“ ソート方向 ” 設定ボックスは選択できません。

(3) ソート方向

解析結果のソート方向を指定します。“ ソートしない (出現順) ” を選択した場合，“ ソート方向 ” 設定ボックスは選択できません。

昇順	昇順でソート
降順	降順でソート（デフォルト）

[ボタン]

(1) [デフォルトの設定に戻す] ボタン

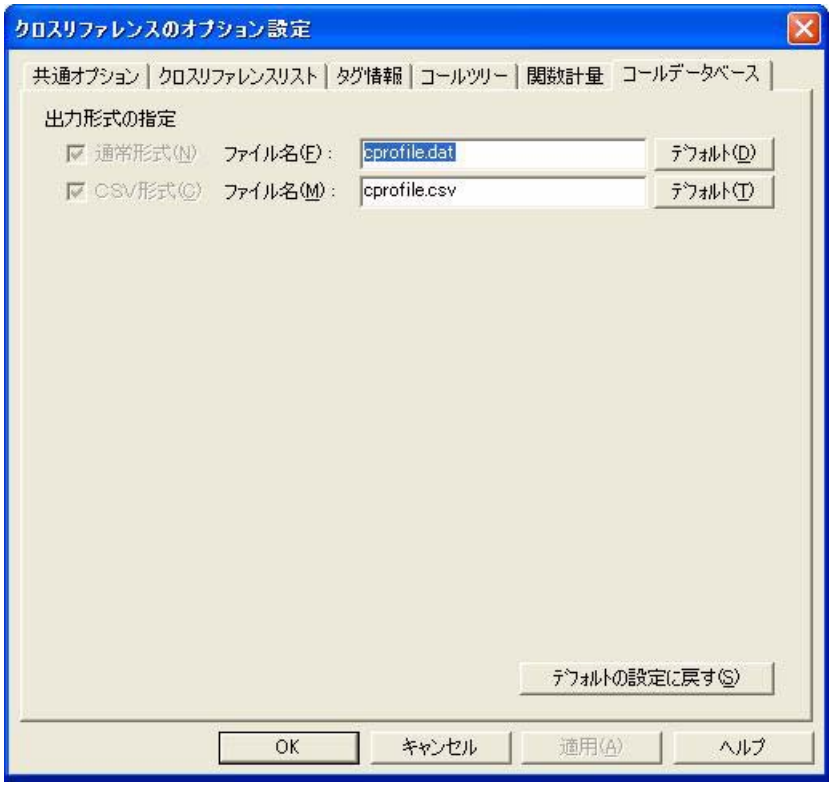
現在のダイアログの設定をデフォルトの状態に戻します。

ただし，[クロスリファレンス] タブで“ すべての解析結果ファイルを出力する ”をチェックしている場合，“ 通常形式 ” と “ CSV 形式 ” はオンのままとなります。

[コールデータベース]

ある関数が，どの関数を何回呼び出しているかの解析に関するオプション設定を行います。

図 12 － 8 [クロスリファレンスのオプション設定] ダイアログ：[コールデータベース] タブ



(1) 出力形式の指定

“ 通常形式 ”をチェックされている場合は，コール・データベースの解析結果がテキスト形式で出力されます。デフォルトでは，チェックされていません。

“ ファイル名： ”エディット・ボックスにより通常形式の解析結果を出力するファイルを指定します。デフォルトは cprofile.dat です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。“ 通常形式 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

“ CSV 形式 ” をチェックされている場合は，コール・データベースの解析結果が CSV 形式で出力されます。デフォルトでは，チェックされていません。

“ ファイル名： ”エディット・ボックスにより CSV 形式の解析結果を出力するファイルを指定します。デフォルトは cprofile.csv です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。

“ CSV 形式 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

ただし，[クロスリファレンス] タブで “ すべての解析結果ファイルを出力する ” をチェックしている場合，“ 通常形式 ” と “ CSV 形式 ” はオンのままとなります。

[ボタン]

(1) **[デフォルトの設定に戻す] ボタン**

現在のダイアログの設定をデフォルトの状態に戻します。

ただし，[クロスリファレンス] タブで“ すべての解析結果ファイルを出力する ”をチェックしている場合，“ 通常形式 ” と “ CSV 形式 ” はオンのままとなります。

12.6 出力形式

この節では，各出力形式の詳細について説明します。

12.6.1 クロス・リファレンス

cxref は，ファイルごとに，そのファイル内で使用されている変数，および関数のクロス・リファレンス情報を出力します。出力先は“ 標準出力（デフォルト）”，または“ テキスト・ファイル ”です。ファイルに出力する場合，デフォルトの出力ファイル名は“ cxref ”です。

図 12 － 9 クロス・リファレンス出力例（cxref）

```
[コマンド入力：cxref -x apli.c]

**** apli.c
 G V NULL    20 30 43 90 91 199 204 205 235
 G F combine #163 187 190
 G F delete  #216 257
 G V deleted #22 203 220 222
        ...
 L V printtree:depth #232 236 242
 G F removeitem #118 178 209
 G F restore #182 208 212
 G V root #20 42 113 115 115 221 223 224 224 224 261
        ...
```

識別子のアルファベット順に出力されます。各行の左から順に，4 種類の情報が出力されます。

(1) リンケージと記憶クラス

次の記号で示します。

G	外部リンケージを持つ静的外部変数，関数
L	内部リンケージを持つ静的外部変数，関数，関数内静的変数
?	不明

(2) 種別

次の記号で示します。

F	関数
V	変数
?	不明

(3) 識別子名

関数名，または変数名そのもの。

ただし，関数内で定義された変数については，名前の重複の可能性があるため，“ 関数名：変数名 ”の形式で示します。

(4) 行番号

定義行番号，および参照行番号が，次の記号付きで列記されます。

! 行番号	宣言行
# 行番号	定義行
? 行番号	宣言・定義であるか，参照であるか不明
記号なし	参照行

12.6.2 タグ情報

cxref は，変数，および関数について，その定義ファイル名と行番号の情報（タグ・ジャンプ情報）を出力します。出力先は“ 標準出力（デフォルト）”，または“ テキスト・ファイル ”です。ファイルに出力する場合，デフォルトの出力ファイル名は“ ctags ”です。

図 12 － 10 タグ情報出力例（cxref）

```
[コマンド入力：cxref -t apli.c]

 apli.c       163    G F combine
 apli.c       216    G F delete
 apli.c       22     G V deleted
 apli.c       194    G F deletesub
 apli.c       22     G V done
 apli.c       108    G F insert
 apli.c       54     G F insertitem
 apli.c       86     G F insertsub
 apli.c       21     G V key
        ...
```

識別子のアルファベット順に出力されます。各行の左から順に，5 種類の情報が出力されます

(1) ファイル名

変数，または関数の定義が記述されているファイル名を示します。

(2) 行番号

変数，または関数の定義の記述位置を示します。

(3) リンケージと記憶クラス

次の記号で示します。

G	外部リンケージを持つ静的外部変数，関数
L	内部リンケージを持つ静的外部変数，関数，関数内静的変数
?	不明

(4) 種別

次の記号で示します。

F	関数
V	変数
?	不明

(5) 識別子名

関数名，または変数名そのもの。

ただし，関数内で定義された変数については，名前の重複の可能性があるため，“ 関数名：変数名 ” の形式で示します。

12.6.3 コール・ツリー

cxref は，-c などのコール・ツリー情報出力オプションを指定すると，“ ある関数がどの関数を呼び出しているか ” を，ツリー状に出力します。出力ファイルの形式は，テキスト形式，または CSV 形式です。主な情報をそのまま参照する場合はテキスト形式に，詳細な情報を表にして参照する場合は CSV 形式に出力すると便利です。

(1) テキスト形式の出力例

-c オプションを指定すると，コール・ツリーがテキスト形式で出力されます。デフォルトの出力ファイル名は “ ccalltre.lst ” です。

テキスト形式の出力は，次のようになります。

図 12 － 11 コール・ツリーのテキスト形式出力例（cxref）

[コマンド入力：cxref -c apli.c]

```
1       @newpage
2       |---malloc?
3       |---printf?
4       +---exit?
5       @search
6       @insertitem
7       @split
8       |---newpage...(1)
9       |---insertitem...(6)
10      +---insertitem...(6)
11      @insertsub
12      |---insertsub*
13      |---insertitem...(6)
14      +---split...(7)
         ...
```

- 処理対象の関数群が，ツリー状に出力されます。
- ツリーのルートとなる関数名の先頭には ‘ @ ’ が付きます。
- ツリーには，提供ライブラリの関数も含まれます。
- 関数名の後ろに表示される記号の意味は，次のとおりです。

?	処理対象のファイル中に定義がない関数であることを示します。
...（数値）	一度出力されているため，以降の出力が省略されたことを示し，数値は，一度目の出力時の行数を示します。 定義されているソース・ファイルを示します。
*	自分自身を呼び出す再帰呼び出し関数のため，以降の出力が中止されたことを示します。

(2) CSV 形式の出力例

-cc オプションを指定すると，コール・ツリーが CSV 形式で出力されます。CSV 形式のファイルは，Microsoft Excel などの表計算ソフトで読み込み可能です。デフォルトの出力ファイル名は “ ccalltre.csv ” です。

CSV 形式の出力は，次のようになります。

図 12 － 12 コール・ツリーの CSV 形式出力例（cxref）

```
[ コマンド入力：cxref -cc apli.c]

[SrcFileList]
No,SrcFileName,FilePath
1,apli.c,

[Funcs]
No,FuncName,SrcFileNo,LineNo,Ret1,Arg1,Ret2,Arg2
1,free,0,0,,,,
2,main,1,248,int,(),,
3,scanf,0,0,,,,
4,delete,1,217,void,(void),,
5,search,1,38,void,(void),,
        ...
[Calltree]
No,FuncNo,FuncAttr,TopFlg,ElimNo,ChildPtr,ChildCnt,RefFileNo,RefLine
1,8,0,1,0,1,3,0,0
2,7,0x21,0,0,0,0,1,30
3,12,0x21,0,0,0,0,1,31
4,9,0x21,0,0,0,0,1,32
[ChildFuncs]
No,CalltreeNo
1,2
2,3
3,4
4,8
5,9
6,10
7,12
8,13
9,14
10,16
        ...
```

【出力内容の説明】

(a) [SrcFileList]

プログラムで使用されている関数の定義されているソース・ファイル名が出力されます。

FileName	ソース・ファイル名。
FilePath	ソース・ファイルのパス。入力したファイルにパスを指定した場合にのみ出力されます。

(b) [Funcs]

プログラムで使用されているすべての関数が出力されます。

FuncName	関数名。
SrcFileNo	ソース・ファイル番号。[SrcFileList] の “ No ” の値を用いて，その関数が定義されているソース・ファイルを示します。
LineNo	行番号。ソース・ファイル中で，その関数の定義が始まる行を示します。
Ret1,Ret2	関数の戻り値。解析できなかった場合は，何も出力されません。
Arg1,Arg2	関数の引数。解析できなかった場合は，何も出力されません。

(c) [Calltree]

コール・ツリーが出力されます。

FuncNo	関数番号。[Funcs] の “ No ” の値を用いて，その関数を示します。
FuncAttr	関数属性。ツリーの属性を，次の数値の組み合わせで示す。属性がない場合，0 が出力されます。 0x0001 プログラム記述がない。 0x0002 再帰関数である。 0x0004 以降のツリーの出力を省略する。 0x0008 ソース・ファイル名，記述開始行を出力する。 0x0010 戻り値，引数を出力する。 0x0020 参照情報を出力する。
TopFlag	トップ・フラグ。その関数がツリーのルートの場合，1 が出力され，ルートでない場合，0 が出力されます。
ElimNo	以前出力された際のツリー番号。その関数が “ FuncAttr ” で “ 0x0004 ” に該当する場合，その関数が以前出力されたツリーを，[Calltree] の “ No ” で示す。 該当しない場合，0 が出力されます。
ChildPtr	子関数表示の開始位置。[ChildFuncs] で，その関数の最初の子関数が出力される位置を，[ChildFuncs] の “ No ” で示します。
ChildCnt	子関数の数。[ChildFunc] に登録された子関数の数を示す。子関数がない場合，0 が出力されます。

(d) [ChildFuncs]

子関数情報として，その子関数が存在するツリーが出力されます。

CallTreeNo	ツリー番号。[Calltree] の “ No ” の値を用いて，その子関数が存在するツリーを示します。

12.6.4 関数計量

cxref は，-m などの関数計量情報出力オプションを指定すると，関数単位の情報を出力します。出力ファイルの形式は，テキスト形式，または CSV 形式です。主な情報をそのまま参照する場合はテキスト形式に，詳細な情報を表にして参照する場合は CSV 形式に出力すると便利です。

(1) テキスト形式の出力例

-m オプションを指定すると，関数計量がテキスト形式で出力されます。デフォルトの出力ファイル名は“ cmeasure.lst ”です。テキスト形式の出力は，次のようになります。

図 12 － 13 関数計量のテキスト形式出力例（cxref）

[コマンド入力：`cxref -m apli.c`]

```
            Flie          Line    Called
newpage     apli.c         27          2
search      apli.c         38          1
insertitem  apli.c         55          3
split       apli.c         68          1
insertsub   apli.c         87          2
insert      apli.c        109          1
removeitem  apli.c        119          2
moveright   apli.c        128          1
movelefta   pli.c         146          1
combin      apli.c        164          2
restore     apli.c        183          2
deletesub   apli.c        195          3
delete      apli.c        217          1
printtree   apli.c        231          3
main        apli.c        248          0
          ...
```

【出力内容の説明】

(a) Flie

ファイル名。その関数が定義されているソース・ファイル名を示します。

(b) Line

開始行。その関数が，ソース・ファイルで何行目に定義されているかを示します。

(c) Called

コール・ヒストグラム。その関数が呼び出される頻度を示します。関数呼び出しの記述 1 つに対し，1 回呼び出されたと想定した頻度が出力されます。

(2) CSV 形式の出力例

-mc オプションを指定すると，関数計量が CSV 形式で出力されます。CSV 形式のファイルは，Microsoft Excel などの表計算ソフトで読み込み可能です。デフォルトの出力ファイル名は “ cmeasure.csv ” です。

CSV 形式の出力は，次のようになります。

図 12 － 14 関数計量の CSV 形式出力例（cxref）

```
[ コマンド入力：cxref -mc apli.c ]
[SrcFileList]
 No,SrcFileName,FilePath
 1,apli.c,

 [Funcs]
 No,FuncName,SrcFileNo,LineNo,Ret1,Arg1,Ret2,Arg2
 1,free,0,0,,,,
 2,main,1,248,int,(),,
 3,scanf,0,0,,,,
 4,delete,1,217,void,(void),,
 5,search,1,38,void,(void),,
        ...
 [Measure]
No,FuncNo,FuncSz,Clk,TClk,Stk,TStk,CalledCnt,StkUp,StkUpPtr,StkUpCnt,ClkUp,
ClkUpPtr,ClkUpCnt,StkDw,StkDwPtr,StkDwCnt,ClkDw,ClkDwPtr,ClkDwCnt
 1,8,64,37,37,12,68,2,68,1,4,496,5,4,12,0,0,37,0,0
 2,5,208,118,118,12,24,1,24,9,1,237,10,1,12,0,0,118,0,0
 3,19,148,71,71,16,72,3,72,11,4,530,15,4,16,0,0,71,0,0
        ...
```

【出力内容の説明】

(a) [SrcFileList]

プログラムで使用されている関数の定義されているソース・ファイル名が出力されます。

FileName	ソース・ファイル名。
FilePath	ソース・ファイルのパス。入力したファイルにパスを指定した場合にのみ出力されます。

(b) [Funcs]

プログラムで使用されているすべての関数が出力されます。

FuncName	関数名。
SrcFileNo	ソース・ファイル番号。[SrcFileList] の “ No ” の値を用いて，その関数が定義されているソース・ファイルを示します。
LineNo	行番号。ソース・ファイル中で，その関数の定義が始まる行を示します。
Ret1,Ret2	関数の戻り値。解析できなかった場合は，何も出力されません。
Arg1,Arg2	関数の引数。解析できなかった場合は，何も出力されません。

(c) [Measure]

関数計量情報が出力されます。

FuncNo	**関数番号。**[Funcs] **の“** No **”の値を用いて，その関数を示します。**
CalledCnt	**コール・ヒストグラム。その関数が呼び出される頻度を示します。関数呼び出しの記述** 1 **つに対し，**1 **回呼び出されたと想定した頻度が出力されます。**

12.6.5 コール・データベース

cxref は，-b などのコール・データベース情報出力オプションを指定すると，“ どの関数がどの関数を何回呼び出しているか ” を出力します。出力ファイルの形式は，テキスト形式，または CSV 形式です。主な情報をそのまま参照する場合はテキスト形式に，詳細な情報を表にして参照する場合は CSV 形式に出力すると便利です。

(1) テキスト形式の出力例

-b オプションを指定すると，コール・データベースがテキスト形式で出力されます。デフォルトの出力ファイル名は “ cprofile.dat ” です。

テキスト形式の出力は，次のようになります。

図 12 － 15 コール・データベースのテキスト形式出力例（cxref）

```
[コマンド入力：cxref -b apli.c]

newpage,apli.c,malloc,0,1
newpage,apli.c,printf,0,1
newpage,apli.c,exit,0,1
split,apli.c,newpage,apli.c,1
split,apli.c,insertitem,apli.c,2
insertsub,apli.c,insertsub,apli.c,1
insertsub,apli.c,insertitem,apli.c,1
insertsub,apli.c,split,apli.c,1
insert,apli.c,insertsub,apli.c,1
insert,apli.c,newpage,apli.c,1
combine,apli.c,removeitem,apli.c,1
combine,apli.c,free,0,1
       ...
```

各行の左から順に，5 種類の情報が出力されます。

(a) 呼び出し元の関数名。

(b) 呼び出し元の関数が定義されているソース・ファイル名。解析できない場合，“ ??? ” が出力されます。

(c) 呼び出し先の関数名。

(d) 呼び出し先の関数が定義されているソース・ファイル名。ライブラリ中の関数の場合，ソース・ファイル名は不明のため，0 が出力されます。

(e) 呼び出し元の関数中で，呼び出し先の関数が呼び出されている回数。

(2) CSV 形式の出力例

-bc オプションを指定すると，変数のサイズ／使用頻度が CSV 形式で出力されます。CSV 形式のファイルは，Microsoft Excel などの表計算ソフトで読み込み可能です。デフォルトの出力ファイル名は，“ cprofile.csv ”です。

CSV 形式の出力は，次のようになります。

図 12 － 16 コール・データベースの CSV 形式出力例（cxref）

```
[ コマンド入力：cxref -bc apli.c]

 [SrcFileList]
No,SrcFileName,FilePath
1,apli.c,

 [Funcs]
No,FuncName,SrcFileNo,LineNo,Ret1,Arg1,Ret2,Arg2
1,free,0,0,,,,
2,main,1,248,int,(),,
3,scanf,0,0,,,,
4,delete,1,217,void,(void),,
5,search,1,38,void,(void),,
        ...
 [CallDataBase]
No,FuncNo,ChildFuncNo,CallCnt
1,8,7,1
2,8,12,1
3,8,9,1
4,11,8,1
5,11,19,2
        ...
```

【出力内容の説明】

(a) [SrcFileList]

プログラムで使用されている関数の定義されているソース・ファイル名が出力されます。

FileName	ソース・ファイル名。
FilePath	ソース・ファイルのパス。-p オプション指定時にのみ出力されます。

(b) [Funcs]

プログラムで使用されているすべての関数が出力されます。

FuncName	関数名。
SrcFileNo	ソース・ファイル番号。[SrcFileList] の “ No ” の値を用いて，その関数が定義されているソース・ファイルを示します。
LineNo	行番号。ソース・ファイル中で，その関数の定義が始まる行を示します。
Ret1,Ret2	関数の戻り値。解析できなかった場合は，何も出力されません。
Arg1,Arg2	関数の引数。解析できなかった場合は，何も出力されません。

(c) [CallDataBase]

コール・データベース情報が出力されます。

FuncNo	**呼び出し元関数番号。**[Funcs] **の“** No **”の値を用いて，呼び出し元の関数を示します。**
ChildFuncNo	**呼び出し先関数番号。**[Funcs] **の“** No **”の値を用いて，呼び出し先の関数を示します。**
CalledCnt	**呼び出されている回数。呼び出し元の関数中で，呼び出し先の関数が呼び出されている回数。**

第 13 章　メモリ・レイアウト視覚化ツール

この章では，メモリ・レイアウト視覚化ツール（rammap）の概要，操作方法，出力ファイルの形式について説明します。

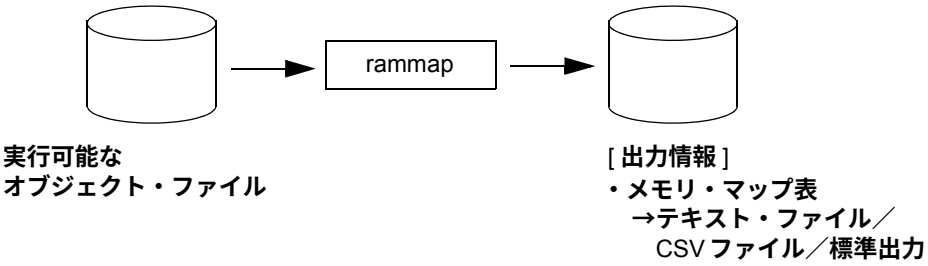
13.1　メモリ・レイアウト視覚化ツールとは

メモリ・レイアウト視覚化ツールとは，作成したロード・モジュールから，変数のメモリ配置情報を読み出し，表示するユーティリティです。

C コンパイラ・パッケージに入っている“ rammap ”がメモリ・レイアウト視覚化ツールです。

変数のメモリ配置情報を，テキスト・ファイルや CSV ファイルとして出力します。

図 13 － 1　rammap における動作の流れ



13.2 入出力

13.2.1 入力ファイル

rammap の入力ファイルは，ld850 が出力した実行可能オブジェクト・ファイル[注]（.out ファイル）です。

注 再リンク可能なオブジェクト・ファイル，または romp850 が出力したファイル（.out ファイル）は除く。

13.2.2 出力情報

rammap が出力する情報は，変数名，サイズ，メモリ配置を示すメモリ・マップです。

(1) メモリ・マップ表

変数名，サイズ，メモリ配置を示すメモリ・マップ表を出力します。

この情報についての詳細は「13.6 出力形式」を参照してください。

13.3 操作方法

この節では，rammap の操作方法について説明します。

13.3.1 コマンド入力による方法

コマンドは，コマンド・プロンプトで次のように入力します。

```
rammap [ オプション ] [ ファイル名 ]
       [ ]    ：[ ] 内は省略できます。
```

13.3.2 PM+ による方法

RAM マップは，PM+ 上で次の 2 つの方法で使用することができます。

(1) [静的性能解析ツール] ダイアログでの使用

RAM マップ設定用ダイアログは，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [静的性能解析ツールの設定] を選択したのち，[RAM マップ] タブをクリック

(2) [オブジェクト解析ツール] ダイアログでの使用

RAM マップ起動用ダイアログは，PM+ でプロジェクトを設定後，次の操作により表示します。

- [ツール] メニュー→ [オブジェクト解析ツールの起動] を選択したのち，[RAM マップ] タブをクリック

メモリ・レイアウト視覚化ツールの起動は， プロジェクトごとに 1 回のため，ファイルごとの設定はありません。

13.4 オプションの種類と機能

次に，rammap のオプションを示します。

-V

rammap の版番号を出力し，終了します。

-all

すべての情報を，テキスト形式，および CSV 形式の各ファイルに出力します。“ -mall ” を指定したのと同じです。

-h

-help

オプションの説明を出力し，終了します。

-m[=*file*]

メモリ・マップ表をテキスト形式で，指定したファイル *file* に出力します。“ *=file* ” を省略した場合，ファイル名は rammap.txt になります。

-mall[=*file*]

メモリ・マップ表をテキスト形式と CSV 形式で，指定したファイルに出力します。ファイル名は *file*.lst，および *file*.csv になります。*file* に拡張子を付けて指定した場合，その拡張子は無視されます。“ *=file* ” を省略した場合，ファイル名は rammap.txt，および rammap.csv になります。

-mc[=*file*]

メモリ・マップ表を CSV 形式で，指定したファイル *file* に出力します。“ *=file* ” を省略した場合，ファイル名は rammap.csv になります。

-mr*range*

メモリ・マップ表を出力する範囲を指定します。

[指定方法]

次の 3 種類の方法で指定します。

(1) 開始アドレスと終了アドレスを指定します。

```
rammap a.out -mr0x10000-0x20000
```

(2) 開始アドレスのみ指定します。この場合，終了アドレスは 0xffffffff となります。

```
rammap a.out -mr0x10000-
```

(3) 終了アドレスのみ指定します。この場合，開始アドレスは 0x0 となります。

```
rammap a.out -mr-0x20000
```

- “ -mr ” と範囲の間に空白を入れないでください。

- アドレスには，8 進数，10 進数，16 進数が指定可能です。

8 進数指定形式	-mr0200000-0400000
10 進数指定形式	-mr65536-131072
16 進数指定形式	-mr0x10000-0x20000

[範囲の複数指定]

複数の範囲を指定することができます。複数指定には -mr オプションを複数指定する方法と，範囲ごとにカンマ（ , ）で区切る方法があります。

(1) -mr を複数指定する方法

```
rammap a.out -mr0x10000-0x20000 -mr0x30000-0x40000
```

(2) カンマで区切る方法

```
rammap a.out -mr0x10000-0x20000,0x30000-0x40000
```

- 指定した範囲の重なりによって，次のようにみなされます。

例 1

```
a ---------- b   c ---------- d
```

この場合，a ～ b，c ～ d の 2 つの範囲を指定したものとみなされます。

例 2

```
a ---------- b
            c ---------- d
```

この場合，a ～ d の 1 つの範囲を指定したものとみなされます。

例 3

```
a ---------- b
      c ---------- d
```

この場合，a ～ d の 1 つの範囲を指定したものとみなされます。

例 4

```
a --------------------------- b
       c ---------- d
```

この場合，a ～ b の 1 つの範囲を指定したものとみなされます。

注意 1 実際のアドレス範囲は，16 バイトに整列されます。

開始アドレスは，指定した値を 16 バイトに丸めた値（0xfffffff0 との論理積）となります。終了アドレスは，指定した値を 16 バイトに丸め，0xF を加算した値となります。

-mr0x10000-0x20000	0x10000 〜 0x2000f
-mr0x10004-	0x10000 〜 0xffffffff
-mr-0x20005	0x0 〜 0x2000f

注意 2　範囲指定が不正な場合，エラー・メッセージが出力され，処理が中断されます。

-mstd

テキスト形式のメモリ・マップ表を標準出力に出力します。

-o *path*

出力ファイルのパス *path* を指定します。このオプションを省略した場合，カレント・パスに出力されます。

@cfile

***cfile* をコマンド・ファイルとして扱います。コマンド・ファイルとは，コマンドに対するオプションやファイル名をコマンド・ラインの引数として指定するのではなくファイルに記述して指定するものです。**

Windows 上では，コマンドに対するオプション指定の文字列の長さに制限があります。数多くのオプションを設定し，オプションを認識しきれない場合などに，コマンド・ファイルを作成し，このオプションを指定してください。

コマンド・ファイルについての詳細は「3.7.2 コマンド・ファイル」を参照してください。

13.5 PM+ での設定

この節では，対象プロジェクトのファイルに対して，rammap のコマンド・オプションを設定するダイアログについて説明します。

13.5.1 [静的性能解析ツール] ダイアログ

[静的性能解析ツール] ダイアログの上部には，次の 2 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

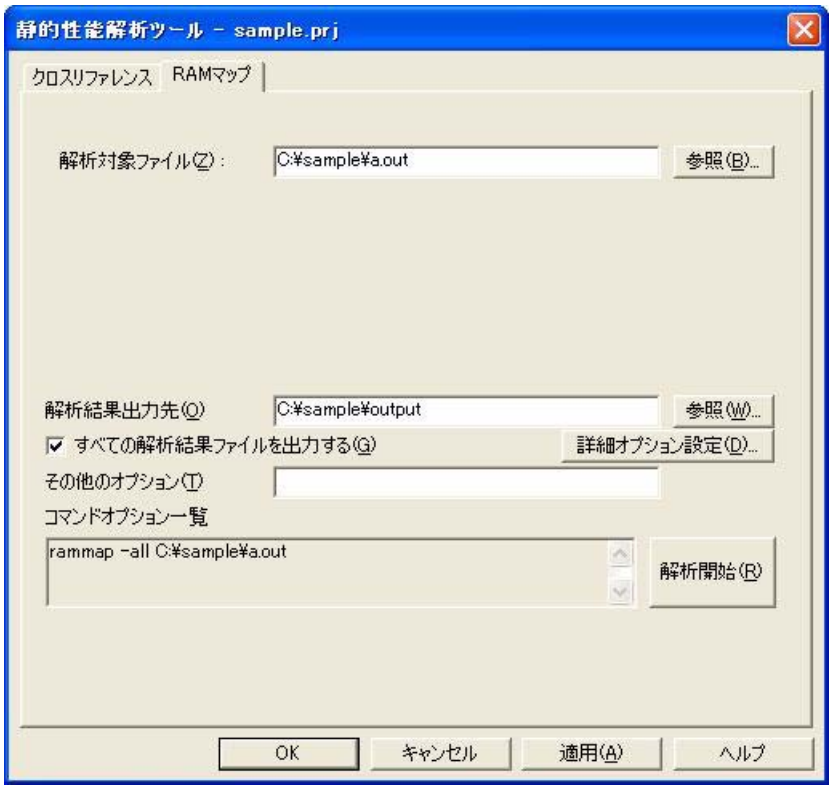
表 13 － 1 [静的性能解析ツール] ダイアログ（rammap）

タブ	内容
[クロスリファレンス]	クロス・リファレンス関するオプションを設定
[RAM マップ]	RAM マップ関するオプションを設定

[RAM マップ]

RAM マップ設定用ダイアログは，プロジェクト・マネージャでプロジェクトを設定後，[ツール] メニュー→ [静的性能解析ツールの設定] でオープンするダイアログの [RAM マップ] タブを選択します。

図 13 － 2 [静的性能解析ツール] ダイアログ：[RAM マップ] タブ



(1) 解析対象ファイル

エディット・ボックスにより解析対象ファイルを指定します。リンカが出力したオブジェクト・ファイル（*.out）を指定してください。デフォルトは，次のようになります。

- オプションを保存している場合は，保存している解析対象ファイルになります。
- オプションを保存していない場合は，プロジェクト・ファイルに登録されているターゲット情報を表示します。

[参照] ボタンにより [フォルダの参照] ダイアログが開き，解析対象ファイルを選択します。選択したファイルは “ 解析対象ファイル ” に表示されます。

[フォルダの参照] ダイアログでは，次の順序でデフォルトの位置が決定されます。

(a) 解析対象ファイルが指定されている場合，解析対象ファイルのフォルダ
(b) [RAM マップ] タブの “ 解析結果出力先 ” のパス
(c) [クロスリファレンス] タブの “ 解析フォルダ ”
(d) インストール・フォルダ

(2) 解析結果出力先

エディット・ボックスにより解析結果出力先（フォルダ）を指定します。デフォルトは，次のようになります。

- オプションを保存している場合は，保存している保存している解析結果出力先になります。
- オプションを保存していない場合は，プロジェクト・ファイルのおかれているフォルダになります。

指定されたフォルダが存在しないとき，[OK] ボタン，または [適用] ボタンにより [フォルダ作成] ダイアログが開き，[OK] ボタンを選択するとフォルダが作成されます。

また，[参照] ボタンにより [フォルダの参照] ダイアログが開き，解析結果出力先を選択します。選択したフォルダは “ 解析結果出力先 ” に表示されます。[フォルダの参照] ダイアログでは，次の順序でデフォルトの位置が決定されます。

(a) 解析対象ファイルが指定されている場合，解析対象ファイルのフォルダ
(b) [RAM マップ] タブの “ 解析対象ファイル ” のパス
(c) [クロスリファレンス] タブの “ 解析フォルダ ”
(d) インストール・フォルダ

(3) すべての解析結果ファイルを出力する

チェックされている場合は，オプション内容をデフォルトの内容で設定します。デフォルトでは，チェックされています。

(4) 詳細オプション設定

[RAM マップのオプション設定] ダイアログが開き，各機能のオプション内容を設定できます（「[共通オプション]」参照）。

(5) その他のオプション

エディット・ボックスにより直接オプションを記述できます。デフォルトでは，空欄です。

(6) コマンドオプション一覧

オプションの一覧が表示されます。デフォルトは，“ rammap -all ” です。

[解析開始] ボタンにより，設定されたオプション内容で解析対象ファイルが解析されます。インストール・フォルダにある解析コマンドが起動されます。解析対象ファイルの指定がない場合，解析ファイルの指定がないことを知らせるメッセージ・ボックスが開きます。

解析の結果により，正常終了，または異常終了を知らせるメッセージ・ボックスが開きます（解析コマンドの出力メッセージは，解析結果出力先のフォルダ，または解析フォルダのファイルrammap.logに出力されます）。

13.5.2 [RAM マップのオプション設定] ダイアログ

[RAM マップのオプション設定] ダイアログの上部には，次の 1 つのタブが表示されています。

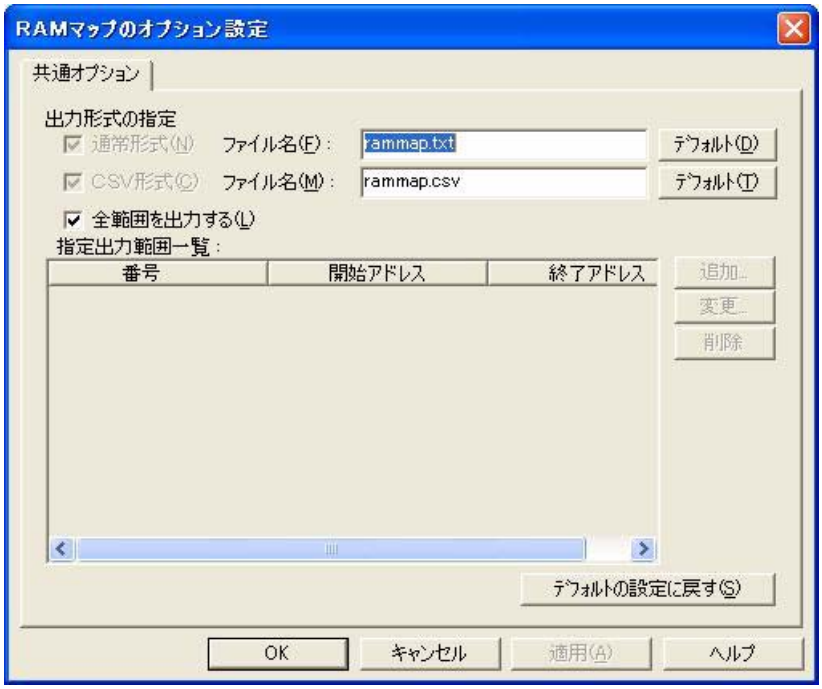
表 13 － 2 [RAM マップのオプション設定] ダイアログ

タブ	内容
[共通オプション]	RAM マップに関するオプション設定

[共通オプション]

RAM マップに関するオプション設定を行います。

図 13 － 3 [RAM マップのオプション設定] ダイアログ：[共通オプション] タブ



(1) 出力形式の指定

“ 通常形式 ” をチェックされている場合は，RAM マップの解析結果がテキスト形式で出力されます。

デフォルトでは，チェックされていません。

“ ファイル名（F）：” エディット・ボックスにより通常形式の解析結果を出力するファイルを指定します。デフォルトは rammap.txt です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。“ 通常形式 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

“ CSV 形式 ”をチェックされている場合は，RAM マップの解析結果が CSV 形式で出力されます。デフォルトでは，チェックされていません。

“ ファイル名（M）：” エディット・ボックスにより CSV 形式の解析結果を出力するファイルを指定します。デフォルトは rammap.csv です。[デフォルト] ボタンを選択することによってもファイル名をデフォルトに設定できます。

“ CSV 形式 ” を選択した場合のみ，この設定ボックス，および [デフォルト] ボタンは選択できます。

ただし，[RAM マップ] タブで “ すべての解析結果ファイルを出力する ” をチェックしている場合，“ 通常形式 ” と “ CSV 形式 ” はオンのままとなります。

(2) 全範囲を出力する

チェックされている場合は，全アドレス範囲の出力を行います。デフォルトでは，チェックされています。チェックされている場合は，“ 指定出力範囲一覧 ”，および [追加] ボタンは選択できません。

(3) 指定出力範囲一覧

リスト・ボックスにより出力範囲が表示されます。デフォルトでは空欄です。

[追加] ボタンにより，開始アドレスと終了アドレスが追加表示され，直接入力はできません。

[追加] ボタンを選択すると [出力範囲の指定] ダイアログが開き，開始アドレスと終了アドレスを指定します。指定した開始アドレスと終了アドレスは一覧に表示されます。[出力範囲の指定] ダイアログでの入力は 16 進数値のみ指定でき，設定可能範囲は 0 ～ 0xffffffff で，デフォルトでは空欄です。指定値が範囲外の場合，指定範囲外であることを知らせるメッセージ・ボックスが開きます。

一覧で，開始アドレスと終了アドレスを選択したのち，[変更] ボタンを選択すると，[出力範囲の指定] ダイアログが開き，そこで選択した開始アドレスと終了アドレスに変更されます。デフォルトでは，指定出力範囲一覧で選択した開始アドレスと終了アドレスです。[出力範囲の指定] ダイアログでの入力は 16 進数値のみ指定でき，設定可能範囲は 0 ～ 0xffffffff で，デフォルトでは空欄です。指定値が範囲外の場合，指定範囲外であることを知らせるメッセージ・ボックスが開き，入力は無効になります。

[削除] ボタンにより，一覧で選択した関数が削除されます。

なお，[変更] ボタン，および [削除] ボタンは，一覧で開始アドレスと終了アドレスを指定した場合のみ選択できます。

ただし，“ 全範囲を出力する ” を選択した場合，この設定ボックス，[追加] ボタン，[変更] ボタン，および [削除] ボタンは選択できません。

[ボタン]

(1) [デフォルトの設定に戻す] ボタン

現在のダイアログの設定をデフォルトの状態に戻します。

ただし，[RAM マップ] タブで “ すべての解析結果ファイルを出力する ” をチェックしている場合，“ 通常形式 ” と “ CSV 形式 ” はオンのままとなります。

13.5.3 [オブジェクト解析ツール] ダイアログ

[オブジェクト解析ツール] ダイアログの上部には，次の 3 つのタブが表示されており，タブの選択により，ダイアログの表示が変わります。

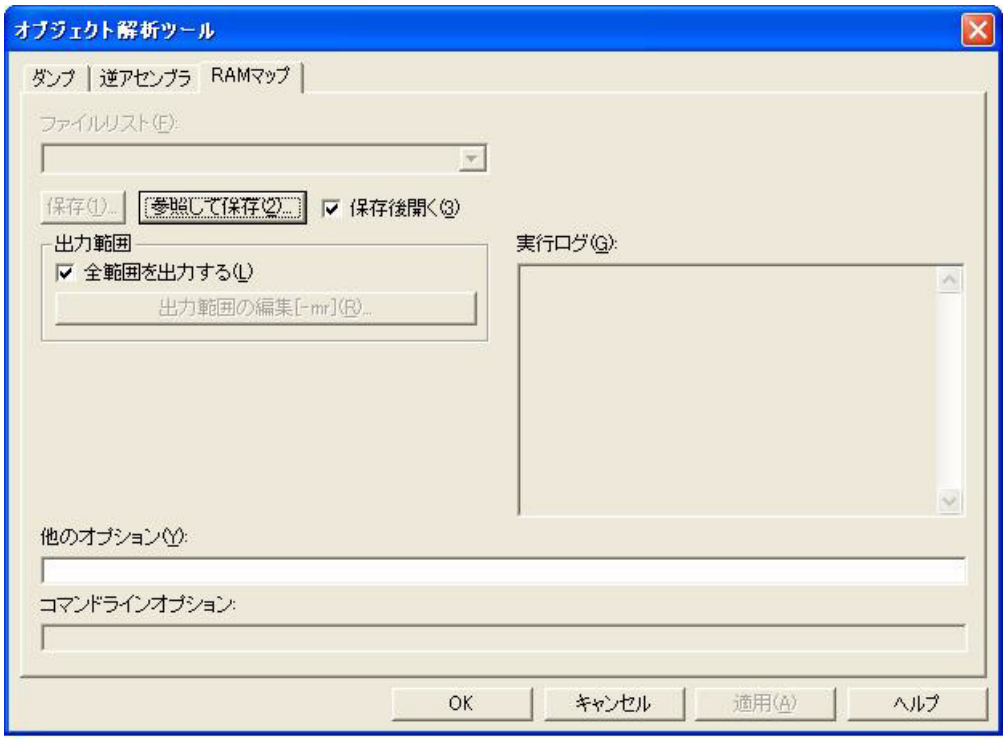
表 13 － 3 [オブジェクト解析ツール] ダイアログ（rammap）

タブ	内容
[ダンプ]	ダンプ・コマンドに関するオプションを設定
[逆アセンブラ]	ディスアセンブラに関するオプションを設定
[RAM マップ]	RAM マップに関するオプションを設定

[RAM マップ]

RAM マップ起動用ダイアログは，プロジェクト・マネージャでプロジェクトを設定後，[ツール] メニュー→ [オブジェクト解析ツールの起動] でオープンするダイアログの [RAM マップ] タブを選択します。

図 13 － 4 [オブジェクト解析ツール] ダイアログ：[RAM マップ] タブ



(1) ファイルリスト

プロジェクトでビルド対象となるオブジェクト・ファイルをドロップダウン・リストに表示します（リストにはパスは表示されません）。

リストとして次のファイルが対象となります。

- リンカの出力ファイル（実行可能オブジェクト・ファイルをビルドするプロジェクトのみ）

なお，このドロップダウン・リストは常に選択不可能です。

[保存] ボタンにより，[名前を付けて保存] ダイアログを開き，“ ファイルリスト ” で指定したオブジェクト・ファイルの RAM マップを保存します。デフォルトでは，“ ファイルリスト ” で指定したオブジェクト・ファイルが存在するフォルダ内に，“ オブジェクト・ファイル名 .txt ” をファイル名として保存します。

なお，[名前を付けて保存] ダイアログにおいて，“ *.csv ” を指定した場合は，CSV 形式で保存されます（それ以外の場合は，テキスト形式で保存）。

ただし，ライブラリをビルドするプロジェクトの場合は，このボタンは無効となります。

[参照して保存] ボタンにより，オブジェクト・ファイルを指定して RAM マップを保存することができます。この場合，[ファイルを開く] ダイアログが開き，オブジェクト・ファイルを指定します。デフォルトでは，前

回の [ファイルを開く] ダイアログで指定したフォルダ，またはプロジェクト・フォルダが表示されます。

[ファイルを開く] ダイアログでオブジェクト・ファイルを指定すると，指定したオブジェクト・ファイルに対する RAM マップをテキスト形式で保存します。

(2) 保存後開く

[保存] ／ [参照して保存] ボタンにより逆アセンブル結果を保存したのち，PM+ 上で設定したエディタ，または関連付けられたアプリケーション（CSV 形式の場合）で保存したファイルを開くがどうかを指定します。

デフォルトでチェックされています。

(3) 出力項目

(a) 全範囲を出力する

チェックすることにより，-mr オプションが無効になります。

デフォルトでチェックされています。

(b) [出力範囲の編集] ボタン

-mr オプションで指定する，出力範囲を編集するための [オプションの編集] ダイアログを開きます。

(4) 実行ログ

RAM マップ実行時のログを表示します。

なお，このエリアは参照用のため，書き込みはできません。

(5) 他のオプション

前記の“ rammap コマンドのオプション ”では設定できないオプションを指定します。このエディット・ボックスにコマンド・ラインと同じ形式で記述します。

なお，rammap コマンドに関するオプションは [オブジェクト解析ツール] ダイアログですべて指定できるため，他のオプションを使用する必要はありません。

(6) コマンドラインオプション

ダイアログで設定したオプションを，コマンド・ライン・オプションで表示します。

なお，このエリアは参照用のため，書き込みはできません。

13.6 出力形式

この節では，出力形式の詳細について説明します。

13.6.1 メモリ・マップ表

rammap は，変数名，サイズ，メモリ配置を示すメモリ・マップ表を出力します。出力先は，“ 標準出力 ”，または “ ファイル ” です。ファイルに出力する場合，出力ファイルの形式は，テキスト形式，または CSV 形式です。主な情報をそのまま参照する場合はテキスト形式に，詳細な情報を表にして参照する場合，CSV 形式に出力すると便利です。

- メモリ・マップ表は，1 行あたり 16 バイトです。
- 変数名は，C 言語ソース・ファイルでの名前を “ name ” とすると，次の形式で表示されます。

外部変数	_name
ファイル内ローカル変数	ファイル名 @_name
関数内静的変数，文字列定	ファイル名 @LL 数字

- サイズは，変数名の後ろに，“（10 進数表記のバイト数）” の形式で表示されます。

(1) テキスト形式の出力例

-m オプションを指定すると，メモリ・マップ表がテキスト形式で出力されます。デフォルトの出力ファイル名は “ rammap.txt ” です。

テキスト形式の出力は，次のようになります。

図 13 － 5 メモリ・マップ表のテキスト形式出力例（rammap）

```
[ コマンド入力：rammap -m a.out ]

Address   +0  +1  +2  +3  +4  +5  +6  +7  +8  +9  +A  +B  +C  +D  +E  +F
-----------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
0x00000000 |
:
0x00FFE000 |crtN.s@__argc(>|crtN.s@__argv(>|               |test.c@LL29(5)-
0x00FFE010 |-->|           |test.c@_svar(4>|_var(4)------->|_gAppName(8)---
0x00FFE020 |-------------->|_c>|           |_tmp(4)------->|_buf(100)------
:
0x00FFE080 |-------------------------------------------------------------->|
0x00FFE090 |_var2(4)------>|_c>|           |crtN.s@__stack(512)------------
:
0x00FFE290 |------------------------------>|
:
0xFFFFFFF0 |
-----------+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
...
```

- 変数名とサイズは，該当するアドレスの先頭に，左詰めで表示されます。
- メモリ配置枠に入りきらない変数名は，入るところまで表示されます。
- 変数名を 1 つも持たない行は，コロン（:）が出力され，省略されます。未使用領域，text 属性セクション，大きな変数の内部が，これに該当します。

(2) CSV 形式の出力例

-mc オプションを指定すると，メモリ・マップ表が CSV 形式で出力されます。CSV 形式のファイルは，Microsoft Excel などの表計算ソフトで読み込み可能です。デフォルトの出力ファイル名は“ rammap.csv ”です。CSV 形式の出力は，次のようになります。

図 13 － 6 メモリ・マップ表の CSV 形式出力例（rammap）

```
[コマンド入力：rammap -mc a.out]

Address,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
0x00000000,,,,,,,,,,,,,,,,
       :
0x00FFE000,crtN.s@__argc(4),,,,crtN.s@__argv(4),,,,,,,,test.c@LL29(5),,,
0x00FFE010,,,,,test.c@_svar(4),,,,_var(4),,,,_gAppName(8),,,
0x00FFE020,,,,,_cInput(1),,,,_tmp(4),,,,_buf(100),,,
       :
0x00FFE090,_var2(4),,,,_c(1),,,,crtN.s@__stack(512),,,,,,,
       :
0xFFFFFFF0,,,,,,,,,,,,,,,,
       ...
```

- 変数名を 1 つも持たない行は，コロン（:）が出力され，省略されます。未使用領域，text 属性セクション，大きな変数の内部が，これに該当します。

第 14 章　スタック見積もりツール

この章では，スタック見積もりツール（stk850）の概要，操作方法，出力形式について説明します。

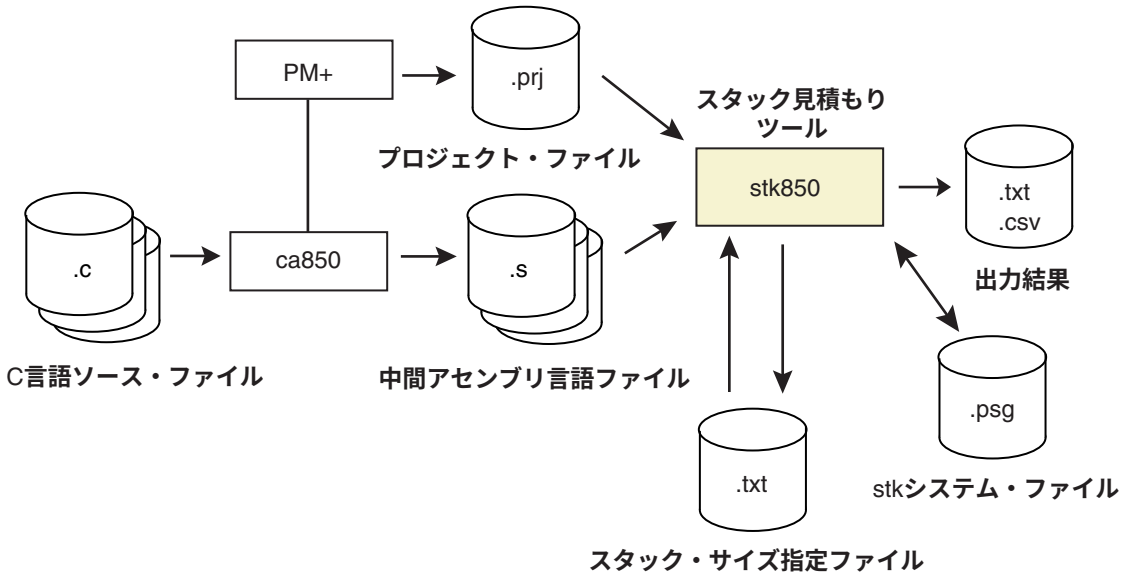
14.1　動作の流れ

スタック見積もりツール“ stk850 ”は，ca850 が出力した中間アセンブリ言語ファイル（.s）を基に，スタック使用量を静的に計算し，ツリー形式で解析結果を表示するツールです。

次のような特長があります。

- PM+ との連携
- スタック・サイズの計算結果を GUI 表示
- スタック見積もりのサイズ変更が容易

図 14 － 1 stk850 における動作の流れ



14.2 入出力ファイル

14.2.1 入力ファイル

stk850 の入力ファイルは次のとおりです。

(1) プロジェクト・ファイル

stk850 は，PM+ のアクティブ・プロジェクトの情報を自動的に読み込みます。

(2) 中間アセンブリ言語ファイル

中間アセンブリ言語ファイルとは，プロジェクト・ファイルに登録された C 言語ソース・ファイルから ca850 が出力した中間アセンブリ言語ソースです。stk850 は，このファイルを基にスタック使用量を解析します。ユーザが，中間アセンブリ言語ファイルを個別に指定することはできません。

(3) スタック・サイズ指定ファイル

stk850 は，ユーザが作成したスタック・サイズ設定情報を，まとめて指定できます。

なお，stk850 は，標準ライブラリ関数のスタック・サイズ指定ファイルとして（stk850 のインストール・フォルダ）\dat850*.txt を使用します。

「14.6.2 スタック・サイズ指定ファイル」を参照。

(4) stk システム・ファイル

「14.6.3 stk システム・ファイル」を参照。

14.2.2 出力ファイル

stk850 の出力ファイルは次のとおりです。

(1) 出力結果ファイル

「14.6.1 出力結果ファイル」を参照。

(2) スタック・サイズ指定ファイル

「14.6.2 スタック・サイズ指定ファイル」を参照。

(3) stk システム・ファイル

「14.6.3 stk システム・ファイル」を参照。

14.3 操作方法

この節では，stk850 の操作方法について説明します。

PM+ からは，次のようにします。

スタック使用量の計算には，中間アセンブリ言語ファイルが必要であるため，次に示す操作で stk850 を起動します。

(1) [コンパイラオプションの設定] ダイアログの [一般] タブで " アセンブラソース [-Fs] " を指定し，リビルドを実行

(2) [ツール] メニュー→ [スタック見積もりツール：STK850 の起動] を選択

stk850 が起動するとメイン・ウインドウが表示され，現在のプロジェクトのスタック使用量が計算されます。各関数については，スタック・サイズの変更，サイズ不明関数一覧の表示などが行えます。

14.4 ウインドウ・リファレンス

stk850 には，次のウインドウ／ダイアログが用意されています。

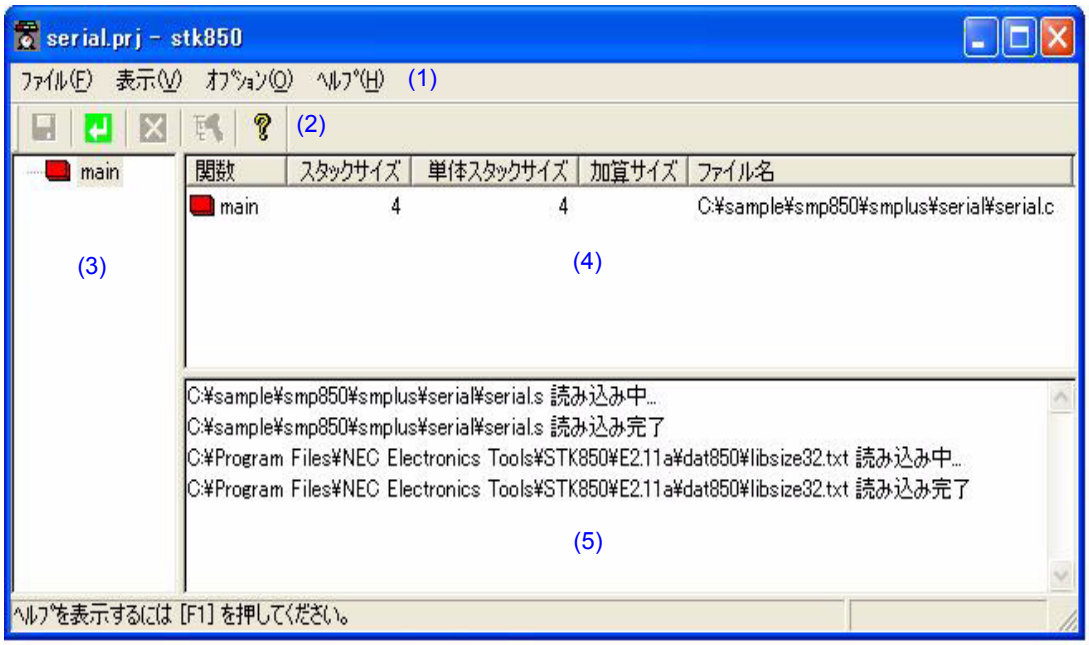
表 14 － 1 stk850 のウインドウ／ダイアログ

ウインドウ／ダイアログ名	内容
メイン・ウインドウ	stk850 起動後，最初に表示されるウインドウです。スタックの使用量を関数ごとに表示します。
[スタックサイズ変更] ダイアログ	関数のスタック・サイズを変更するダイアログです。
[サイズ不明関数・サイズ変更関数一覧] ダイアログ	サイズ不明の関数，およびサイズの変更された関数の一覧を表示するダイアログです。
[stk850 のバージョン情報] ダイアログ	stk850 のバージョンを表示するダイアログです。

メイン・ウインドウ

stk850 起動後，最初に表示されるウインドウです。スタックの使用量を関数ごとに表示します。

図 14 － 2 stk850 メイン・ウィンドウ



(1) メニュー・バー

(a) [ファイル]

プロジェクト・ファイルを開く	選択したプロジェクト・ファイルを読み込み，スタック・サイズを再計算します。
選択した関数の最大経路の保存	選択した関数のスタック・サイズが最大となる経路のスタック・サイズをファイルに保存します。
選択した関数の全経路保存	選択した関数以下の全経路のスタック・サイズをファイルに保存します。
全ルート関数の最大経路の保存	全ルート関数のスタック・サイズが最大となる経路のスタック・サイズをファイルに保存します
全ルート関数の全経路保存	全経路のスタック・サイズをファイルに保存します。
スタックサイズ指定ファイルを開く	選択したスタック・サイズ指定ファイルの情報を読み込みます。
スタックサイズ指定ファイルの保存	スタック・サイズ指定情報をファイルに保存します。
stk850 の終了	stk850 を終了します。

(b) [表示]

スタックサイズ再計算	スタック・サイズを再計算します。
中止	作業を中止します。
アイコンの整列	選択したメニューに従いリスト表示部をソートします。
関数名順	リスト表示部を関数名順でソートします。
アイコン順	リスト表示部をアイコンの色でソートします。
スタックサイズ順	リスト表示部をスタック・サイズ順でソートします。
単体スタックサイズ順	リスト表示部を単体スタック・サイズ順でソートします。
加算サイズ順	リスト表示部を加算サイズ順でソートします。
ファイル名順	リスト表示部をファイル名順でソートします。

(c) [オプション]

[サイズ不明関数・サイズ変更関数一覧] ダイアログ	[サイズ不明関数・サイズ変更関数一覧] ダイアログをオープンします。
[スタックサイズ変更] ダイアログ	[スタックサイズ変更] ダイアログをオープンします。
指定関数を初期値に戻す	指定関数を初期値にします。
全関数を初期値に戻す	全関数を初期値にします。

(d) [ヘルプ]

stk850 のヘルプ	stk850 のヘルプをオープンします。
[stk850 のバージョン情報] ダイアログ	[stk850 のバージョン情報] ダイアログをオープンします。

(2) ツールバー

	[ファイル] メニュー→ [選択した関数の最大経路の保存] 選択した関数のスタック・サイズが最大となる経路をファイルに保存します。
	[表示] メニュー→ [スタックサイズ再計算] スタック・サイズを再計算します。
	[表示] メニュー→ [中止] 作業を中止します。
	[オプション] メニュー→ [[スタックサイズ変更] ダイアログ] [スタックサイズ変更] ダイアログをオープンします。
	[ヘルプ] メニュー→ [stk850 のヘルプ] stk850 のヘルプをオープンします。

(3) ツリー表示部

関数をツリー形式で表示します。

なお，ツリー表示部で関数の変更，並び替えはできません。

関数の状態によりアイコンは，次のようになります（優先順位の高い順）。たとえば，スタック・サイズを変更した関数が呼び出し関数内で最大サイズとなる場合，水色ではなく，赤色になります。

表 14 － 2 stk850 の関数アイコン

アイコン		意味
	赤色	呼び出し関数内で最大サイズとなる関数
	水色	スタック・サイズを変更，または呼び出し関数の追加をしている関数
	緑色	再帰呼び出し関数，または再帰の途中にある関数
	黄色	サイズ未確定関数
	白色	通常関数
	タスク	#pragma rtos_tsk で宣言されたタスク

〔コンテキスト・メニュー〕

[スタックサイズ変更] ダイアログ	選択している関数に対して，[スタックサイズ変更] ダイアログをオープンします。

(4) リスト表示部

ツリー表示部で選択した関数と呼び出す関数をリスト形式で表示します。

なお，リスト表示部で関数の変更はできません。

(a) 関数

関数名を表示します。[関数] をクリックするとリスト表示部を名前順にソートします。関数の種類により次の表示をします。

[*func*]	定義ファイルが不明な場合，関数名を [] で囲み表示します。
file.c#*func*	スタティック関数の場合，パスを含まないファイル名と “ # ” を関数名の前に付けた “ ファイル名 # 関数名 ” で表示します。
*func**	再帰関数の場合，関数名の後ろに “ * ” を付けて表示します。
func&	関数ポインタを用いた間接関数呼び出しを含む場合，関数名の後ろに “ & ” を付けて表示します。

(b) スタックサイズ

呼び出し関数を考慮したスタック・サイズを表示します。スタック・サイズが不明な場合は“？”，限界値を越える場合は“ SIZEOVER ”を表示します。[スタックサイズ] をクリックするとリスト表示部をサイズ順にソートします。

サイズは，（単体スタック・サイズ + 呼び出す関数中最大となるスタック・サイズ + 加算サイズ）で算出されます。再帰関数の場合は，この値が再帰回数倍されます。

(c) 単体スタックサイズ

呼び出し関数を含まない関数の単体スタック・サイズを表示します。スタック・サイズが不明な場合は“？”，限界値を越える場合は“ SIZEOVER ”を表示します。[単体スタックサイズ] をクリックするとリスト表示部をサイズ順にソートします。

(d) 加算サイズ

スタック・サイズに対して加算するサイズの情報を表示します。加算サイズが指定されない関数の場合，空白となります。再帰回数が指定された再帰関数の場合，“ * ”の後に再帰回数を表示します。[加算サイズ] をクリックするとリスト表示部をサイズ順にソートします。

(e) ファイル名

関数が定義されている C 言語ソース・ファイル名を表示します。定義されたファイルがない場合は，空白のままです。[ファイル名] をクリックするとリスト表示部を名前順にソートします。

〔コンテキスト・メニュー〕

[スタックサイズ変更] ダイアログ	選択している関数に対して，[スタックサイズ変更] ダイアログをオープンします。
アイコンの整列	選択したメニューに従いリスト表示部をソートします。
関数名順	リスト表示部を関数名順でソートします。
アイコン順	リスト表示部をアイコンの色でソートします。
スタックサイズ順	リスト表示部をスタック・サイズ順でソートします。
単体スタックサイズ順	リスト表示部を単体スタック・サイズ順でソートします。
加算サイズ順	リスト表示部を加算サイズ順でソートします。
ファイル名順	リスト表示部をファイル名順でソートします。

(5) メッセージ表示部

警告メッセージと進行状況を表示します。

〔コンテキスト・メニュー〕

コピー	選択している部分をクリップ・ボードにコピーします。
クリア	メッセージ表示部をクリアします。

[スタックサイズ変更] ダイアログ

関数のスタック・サイズを変更をするダイアログです。

図 14 － 3 [スタックサイズ変更] ダイアログ

(1) 関数名

選択されている関数名を表示します。

(2) 単体スタックサイズ

呼び出し関数を含まない関数単体でのスタック・サイズを表示します。サイズが不明な場合は“？”を表示します。計算上は“？”はサイズ 0 として扱われます。

(3) 加算サイズ

スタック・サイズに加算する値を入力します。

- 0，正の 10 進数，または“ 0x ”を先頭とした 16 進数で指定します。
- スタック・サイズが不明な（“？”）関数に対して加算サイズを指定した場合，スタック・サイズは“？”からに（単体スタック・サイズ + 呼び出す関数中最大となるスタック・サイズ + 加算サイズ）（再帰関数の場合はその再帰回数倍）に変わります。

(4) 再帰回数

再帰関数の呼び出し回数を指定します。スタック・サイズが指定回数倍のサイズに設定されます。

(5) 呼び出し関数リスト（関数ポインタ向け）

(a) 呼び出し関数一覧

この関数から呼び出す関数一覧を表示します。追加した関数は，関数名の先頭に“＋”を付けた関数名で表示します。

(b) 関数一覧

登録された中間アセンブリ言語ファイルで使用されている関数一覧を表示します。

(c) [追加] ボタン

関数一覧で選択した関数を呼び出し関数一覧に追加します。

(d) [削除] ボタン

呼び出し関数一覧で選択した関数を削除します。実際に呼び出す関数は削除できません。

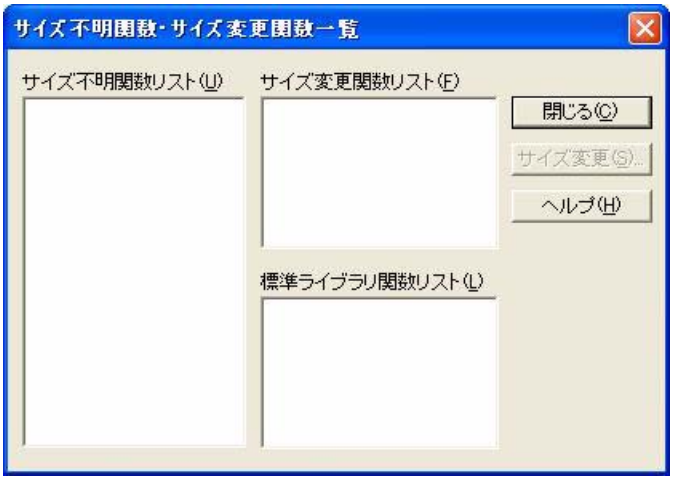
(6) その他のボタン

OK	変更内容を stk システム・ファイルに保存し，ダイアログをクローズします。
キャンセル	変更内容を保存せずにダイアログをクローズします。
初期化	選択した関数の設定をクリアし，初期値に戻します。
ヘルプ	このダイアログのヘルプをオープンします。

[サイズ不明関数・サイズ変更関数一覧] ダイアログ

サイズ不明の関数，およびサイズの変更された関数の一覧を表示するダイアログです。

図 14 － 4 [サイズ不明関数・サイズ変更関数一覧] ダイアログ



(1) サイズ不明関数リスト

スタック・サイズが不明，かつ未確定な関数の一覧を表示します。

スタック・サイズが不明な関数とは，次の関数となります。

- ライブラリ関数
- アセンブリ言語で記述された関数
- 再帰関数
- 関数ポインタを用いた間接関数呼び出しを含む関数

関数がスタティック関数の場合は，パスを含まないファイル名と“＃”を関数名の前に付けた“ファイル名＃関数名”で表示します。定義ファイルが不明な場合は，“[関数名]”の形式で表示します。

(2) サイズ変更関数リスト

スタック・サイズや呼び出し関数の変更をした関数と変更内容の一覧を表示します。

“関数名（スタックサイズ：単体スタックサイズ：変更情報）”の形式で表示します。変更情報は加算サイズ（+ サイズ），再帰回数（* 回数）を表示します。変更情報がないのに追加されている場合は，呼び出し関数の設定を変更しています。

関数がスタティック関数の場合は，パスを含まないファイル名と“＃”を関数名の前に付けた“ファイル名＃関数名”で表示します。定義ファイルが不明な場合は，“[関数名]”の形式で表示します。

(3) 標準ライブラリ関数リスト

登録された中間アセンブリ言語ファイルで使用されている標準ライブラリ関数を表示します。この設定に関しては初期化，または全初期化をしても stk850 が自動で再設定します。標準の設定からサイズを変更すると“サイズ変更関数リスト”に移動します。

(4) その他のボタン

閉じる	このダイアログをクローズします。
サイズ変更	選択した関数で [スタックサイズ変更] ダイアログをオープンします。
ヘルプ	このダイアログのヘルプをオープンします。

[stk850 のバージョン情報] ダイアログ

stk850 のバージョンを表示するダイアログです。

図 14 － 5 [stk850 のバージョン情報] ダイアログ

(1) その他のボタン

OK	このダイアログをクローズします。

14.5 注意事項

この節では，stk850 の注意事項について説明します。

14.5.1 stk850 の限界値

(1) プロジェクト・ファイル関連

プロジェクト・ファイル関連の限界値は次のとおりです。

表 14 － 3 プロジェクト・ファイル関連の限界値

項目	限界値
ファイル数	1
ファイル名の長さ	255
一行の文字数	5,119
ファイルの行数	2,097,152

(2) 中間アセンブリ言語ソース関連

中間アセンブリ言語ソース関連の限界値は次のとおりです。

表 14 － 4 中間アセンブリ言語ソース関連の限界値

項目	限界値
ファイル数	2,048
ファイル名の長さ	255
シンボル名の長さ	1,022
一行の文字数	5,119
ファイルの行数	65,535

(3) スタック・サイズ指定ファイル関連

スタック・サイズ指定ファイル関連の限界値は次のとおりです。

表 14 － 5 スタック・サイズ指定ファイル関連の限界値

項目	限界値
ファイル数（同時読み込み数）	1
ファイル名の長さ	255
一行の文字数	5,119
一行に指定できる関数の数	1
関数に指定できる加算情報の数	1 ～ 1,024
ファイルの行数	32,767

表 14 － 5 スタック・サイズ指定ファイル関連の限界値

項目	限界値
登録できる関数の数	32,767

(4) 出力ファイル関連

出力ファイル関連の限界値は次のとおりです。

表 14 － 6 出力ファイル関連の限界値

項目	限界値
ファイル名の長さ	255
一行の長さ	5,119
ファイルの行数	32,767

(5) スタック・サイズ関連

スタック・サイズ関連の限界値は次のとおりです。

表 14 － 7 スタック・サイズ関連の限界値

項目	限界値
加算情報指定数	1 ～ 1,024
加算サイズ指定	0 ～ 2,147,483,647
再帰回数（スタック・サイズと乗算後）	0 ～ 2,147,483,647
スタック・サイズ限界	2,147,483,647

(6) メッセージ表示部

メッセージ表示部の限界値は次のとおりです。

表 14 － 8 メッセージ表示部の限界値

項目	限界値
一行の文字数	5,119
行数	32,767

14.6 出力形式

この節では，次の出力ファイルの形式について説明します。

- 出力結果ファイル
- スタック・サイズ指定ファイル
- stk システム・ファイル

14.6.1 出力結果ファイル

スタック・サイズの計算結果は，全経路と最大経路の二種類から選択できます。

- メイン・ウインドウで [ファイル]-[選択した関数の全経路保存] を選択すると，選択した関数以下の全関数のスタック・サイズ情報が出力されます。
- メイン・ウインドウで [ファイル] -[選択した関数の最大経路の保存] を選択すると，選択した関数以下で，スタック・サイズが最大となる経路のみが出力されます。

なお，保存時にファイルの種類を選択することでテキスト形式か，CSV 形式の保存ができます。デフォルトはテキスト形式になります。

(1) テキスト形式

テキスト形式で出力する場合，コールツリー形式で出力されます。

呼び出し関数を複数持つ場合 “ +-- ” で関数間を表し，呼び出し関数を 1 つしか持たない場合 “ --- ” で表します。一行の最大文字数が 5,119 文字で，5,119 文字を越えた部分は改行されます。また，出力の最大行数は，32,767 行であり，これを越えた場合は，警告メッセージを表示し，出力が中断されます。

関数の情報は次の形式で表示されます。[ ] 付きのパラメータは不要な場合には表示されません。

- 関数名 [付属情報] (スタックサイズ , 単体スタックサイズ [, 加算情報])

表 14 － 9 各パラメータの説明

パラメータ	内容
関数名	C 言語ソースで定義された関数名。C 言語ソース・ファイルがない場合，“ [シンボル] ” の形式で出力されます 。シンボルは，アセンブリ言語ソース中のシンボル名から先頭の “ _ ” を取ったシンボル名になります。 また，関数がスタティック関数の場合，パスを含まないファイル名と “ # ” を関数名の前に付けた “ ファイル名 # 関数名 ” で表示されます。
付属情報	サイズが未確定の場合の付属情報は次の種類があります。 ‘ * ’ : 再帰関数 ‘ & ’ : 関数ポインタを持つ関数 関数ポインタを用いた間接関数呼び出しを含む関数は，[スタックサイズ変更] ダイアログの “ 呼び出し関数リスト（関数ポインタ向け） ” で指定します。指定した関数は，通常の呼び出し関数と同様に扱われます。
スタックサイズ	関数が使用する最大となるスタック・サイズが表示されます。 サイズの計算は，「(関数単体のスタック・サイズ＋呼び出す関数中最大となるスタック・サイズ + 加算サイズ）×再帰回数」で計算されます。サイズが不明な場合は “ ? ” を，限界値を越えた場合は “ SIZEOVER ” を表示します。

表 14 － 9 各パラメータの説明

パラメータ	内容
単体スタックサイズ	呼び出し関数のスタック・サイズを考慮しない，関数単体でのスタック・サイズ。サイズが不明な関数の場合“？”が表示されます。
加算情報	スタック・サイズを変更するための情報。 加算サイズ（+size）と再帰回数（*time）の片方，または両方が表示されます。

表 14 － 10 加算情報の表示形式と内容

表示形式	内容
+size	size は，［スタックサイズ変更］ダイアログの“加算サイズ”，またはスタック・サイズ指定ファイルで“ADD=”指定されたサイズです。10 進数で表示されます。
*time	再帰関数に対して指定された加算情報。 time は，［スタックサイズ変更］ダイアログの“再帰回数”，またはスタック・サイズ指定ファイルで指定された再帰回数です。10 進数で表示されます。

(a) 全経路出力例

```
main(800, 80)+--sub1(720, 240)---sub2(480, 200)+--sub3(280, 280)
             |                                  +--sub31(160, 160)
             +--sub11*(500, 30, +20*10)---sub11*(500, 30, +20*10)
             +--sub12&(400, 200)---sub21(200, 200)
             +--f.c#sub13(300, 250, +50)
             +--sub14(50, ?, +50)
             +--sub15(?, ?)
```

(b) 最大経路出力例

```
main(800, 80)---sub1(720, 240)---sub2(480, 200)---sub3(280, 280)
```

(2) CSV 形式

CSV 形式で出力する場合，出力内容はテキスト形式と同じです。関数に関する情報と関数間の情報を“ , ”で区切り，経路を埋めた形で出力されます。

ただし，テキスト形式と異なり，一行の最大文字数や最大行数を越える場合，改行せずに警告メッセージを出力し，ファイルの出力が中断されます。

関数に関する情報は次のフォーマットで出力されます。

- 関数名 , スタックサイズ , 単体スタックサイズ , 加算情報

各パラメータに関しては，テキスト形式と同じ内容です。

ただし，加算情報を持たない関数も加算情報“ 0 ”が出力されます。

(a) 全経路出力例

```
関数名 , スタックサイズ , 単体スタックサイズ , 加算情報
main, 800, 80, 0, sub1, 720, 240, 0, sub2, 480, 200, 0, sub3, 280, 280, 0
main, 800, 80, 0, sub1, 720, 240, 0, sub2, 480, 200, 0, sub31, 160, 160, 0
main, 800, 80, 0, sub11*, 500, 30, +20*10, sub11*, 500, 30, +20*10
main, 800, 80, 0, sub12&, 400, 200, 0, sub21, 200, 200, 0
main, 800, 80, 0, f.c#sub13, 300, 250, +50
main, 800, 80, 0, sub14, 50, ?, +50
main, 800, 80, 0, sub15, ?, ?, 0
```

(b) 最大経路出力例

```
関数名 , スタックサイズ , 単体スタックサイズ , 加算情報
main, 800, 80, 0, sub1, 720, 240, 0, sub2, 480, 200, 0, sub3, 280, 280, 0
```

14.6.2 スタック・サイズ指定ファイル

スタック・サイズ指定ファイルとは，正確にスタック・サイズを調べることができない関数（アセンブリ言語で作られた関数，ライブラリ関数，関数ポインタを含む関数，再帰関数）に対してスタック・サイズ設定情報を，まとめて指定するためのテキスト・ファイル（拡張子“ .txt ”）です。

(1) スタック・サイズ指定ファイルの読み込み方法

スタック・サイズ指定ファイルの読み込み方法は，次のとおりです。

- [ファイル] メニュー→ [スタックサイズ指定ファイルを開く] を選択

ファイルを読み込むと，情報を反映し，スタック・サイズの再計算を行います。また，stk システム・ファイルに情報を追加します。すでに設定済みの関数に対しての指定があった場合は，読み込んだ情報を使います。

(2) スタック・サイズ指定ファイルの保存方法

スタック・サイズ指定ファイルの保存方法は，次のとおりです。

- [ファイル] メニュー→ [スタックサイズ指定ファイルの保存] を選択

設定済みの情報と未確定の関数の情報を保存します。未設定の関数は，関数名のみでサイズ設定情報はありません。

(3) スタック・サイズ指定ファイルのフォーマット

スタック・サイズ指定ファイルのフォーマットは，関数名と，サイズ設定情報を“ , ”で区切り指定します。スペース（半角スペース，またはタブ）は区切りとして扱いません。スペースは“ , ”の前後のみ無視し，サイズ設定情報の“ = ”の前後には使うことはできません。

ファイルは，一行につき一関数のサイズ設定情報を指定します。行の先頭に“ # ”がある場合は，コメント行とし，空白行も無視します。関数名のみを指定した場合は何も変更をしません。

- 関数名 [, ADD=size][, RECTIME=time][, CALL=func]

表 14 － 11 各パラメータの説明

パラメータ	内容
関数名	スタック・サイズを変更する関数名を，stk850 が表示する関数名，またはアセンブリ言語ソース中で定義されたシンボル名で指定します 。 表示する関数名とは，C 言語ソースで定義された関数名になります。定義した C 言語ソース・ファイルがない場合（アセンブリ言語ソースで定義されている場合，またはライブラリ関数の場合）は，関数名を“ [] ”で囲んだ名前になります 。関数がスタティック関数の場合は，パスを含まないファイル名と“ # ”を関数名の前に付けた“ ファイル名 # 関数名 ”となります。
ADD=*size*	スタック・サイズを変更するためのサイズ設定情報です。 *size* は，10 進数，または“ 0x ”から始まる 16 進数を指定します。指定した値をスタック・サイズに加算します。サイズ不明関数に対して 0 を指定した場合は，スタック・サイズ“ 0 ”の設定済みの関数として表示します。
RECTIME=*time*	再帰関数に対して回数を指定するためのサイズ設定情報です。 *time* は，10 進数または，“ 0x ”から始まる 16 進数を指定します。指定した値で，スタック・サイズを *time* 倍にします。

表 14 － 11 各パラメータの説明

パラメータ	内容
CALL=*func*	呼び出し関数を追加するためのサイズ設定情報です 。 *func* は，関数名を指定します。関数名の書式は，スタック・サイズを変更する関数名と同じです。指定した関数は，呼び出し関数に追加されます。最大 1024 まで指定できます。スタック・サイズとしては，呼び出し関数の中でスタック・サイズが最大の関数だけが考慮されます。

以下はスタック・サイズ指定ファイルのサンプルと，使用前／後の出力イメージ になります。

(a) スタック・サイズ指定ファイルの記述例

```
# sample.txt
# "_flib”のスタック・サイズを 50 にする。
[flib], ADD=50
# "_flib_zero”を 0 で設定済みとする。表示の変更が目的。
[flib_zero], ADD=0
# サイズの変更（16 進数で指定）と，再帰関数の呼び出し回数の指定（3 回)。
sub2, ADD=0xa, RECTIME=3
# 関数ポインタの呼び出し関数に“ sub4() ”と“ _flib ”を追加
sub3, CALL=sub.c#sub4, CALL=[flib]
# 何も設定しない記述。
sub.c#sub4
```

(b) スタック・サイズ指定ファイル使用前の出力イメージ

```
main(310, 100)+--sub1(210, 200)---sub2*(10, 10)---sub2*(10, 10)
              +--sub3&(200, 200)---[flib_zero](?, ?)
              +---sub.c#sub4(50, 50)---[flib](?, ?)
task1(100, 100)---[_cre_tsk](0, 0)
```

(c) スタック・サイズ指定ファイル（sample.txt）使用後の出力イメージ

```
main(400, 100)+--sub3&(300, 200)+--sub.c#sub4(100, 50)---[flib](50, ?, +50)
              |                  +--[flib](50, ?, +50)
              |                  +--[flib_zero](0, ?, +0)
              +--sub1(260, 200)---sub2*(60, 10, +10*3)---sub2*(60, 10, +10*3)
              +--sub.c#sub4(100, 50)---[flib](50, ?, +50)
task1(172, 100, +72)---[_cre_tsk](0, 0)
```

サイズ設定情報に次のような指定があった場合，不正な指定として扱います。ファイルの行数が限界値を越えた場合は，エラー・メッセージをダイアログとメッセージ表示部に出力し，読み込みを中止します。

- 各限界値を越えた場合（ファイルの行数，一行の文字数，関数名の長さ，サイズ設定情報の数，スタック・サイズの値）
- 使用していない関数名に対する指定
- ファイル内で，同じ関数に対する指定が複数あった場合
- ADD，RECTIME，CALL に対して引数の指定がない
- ADD，RECTIME，CALL 以外のサイズ設定情報
- ADD，RECTIME に不正な引数（10 進数または，“ 0x ” から始まる 16 進数以外）を指定
- CALL に不正な引数（不正な関数名）を指定
- RECTIME を再帰関数でない関数に指定
- 1 つの CALL に複数の関数を指定

14.6.3 stk システム・ファイル

stk システム・ファイルとは，スタック・サイズを変更した情報を保存するファイルです。

ファイル名は，“ プロジェクト・ファイル名 .psg ” となります。読み込み／保存は，stk850 が行います 。stk システム・ファイルの形式は，スタック・サイズ指定ファイルと同じです。

ただし，保存する情報は，サイズが変更された関数のみとなります。

注意 stk システム・ファイルはテキスト・ファイルですが，変更しないでください。

付録 A　オブジェクト・ファイルの形式

ここでは，CA850 において用いられるオブジェクト・ファイルの形式について説明します。

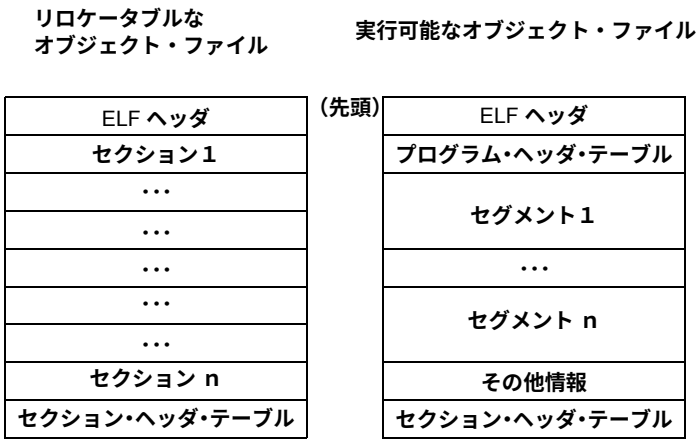
A.1　オブジェクト・ファイルの構造

CA850 において用いられるオブジェクト・ファイルの形式は，標準的なオブジェクト・ファイルの形式の 1 つである ELF 形式に準拠しています。

この形式におけるオブジェクト・ファイルの構造は，リロケータブルなオブジェクト・ファイルと実行可能なオブジェクト・ファイルにおいて多少異なっています（図 A － 1 参照）。リロケータブルなオブジェクト・ファイルは，実行可能なオブジェクト・ファイルを作成するうえで必要とされる情報を持っており，実行可能なオブジェクト・ファイルは，オブジェクト・ファイルを実行するうえで必要とされる情報を持っています。

このあとの各節では，この形式のオブジェクト・ファイルの構成要素である ELF ヘッダ，プログラム・ヘッダ・テーブル，セクション・ヘッダ・テーブル，セクション，およびセグメントについて説明します。

図 A － 1　オブジェクト・ファイルの構造



A.2 ELF ヘッダ

この節では，オブジェクト・ファイル形式の構成要素である ELF ヘッダについて説明します。

ELF ヘッダは，オブジェクト・ファイルの先頭に位置し，そのオブジェクト・ファイルを解釈するための情報やそのオブジェクト・ファイルに含まれる他の構成要素をアクセスするための情報（図 A － 1 参照）を持っています。

表 A － 1 ELF ヘッダの構成要素とその意味

構成要素	意味
ident[CLASS]	オブジェクト・ファイルのクラス
ident[DATA]	オブジェクト・ファイル内のデータのバイト・オーダ（ビッグ・エンディアンである場合 2MSB ／リトル・エンディアンである場合 2LSB）
type	オブジェクト・ファイルの種類
machine	オブジェクト・ファイルの対象とするプロセッサ
version	オブジェクト・ファイル形式の版番号
entry	エントリ・ポイント・アドレス
phoff	プログラム・ヘッダ・テーブルのファイル内オフセット
shoff	セクション・ヘッダ・テーブルのファイル内オフセット
flags	オブジェクト・ファイルが動作するプロセッサに固有なフラグ
ehsize	本 ELF ヘッダのバイト・サイズ
phentsize	プログラム・ヘッダ・テーブルのエントリのサイズ
phnum	プログラム・ヘッダ・テーブルのエントリの数
shentsize	セクション・ヘッダ・テーブルのエントリのサイズ
shnum	セクション・ヘッダ・テーブルのエントリの数
shstrndx	セクション名を保持しているストリング・テーブル .shstrtab のセクション・ヘッダ・テーブル・インデックス

A.3 プログラム・ヘッダ・テーブル

この節では，オブジェクト・ファイル形式の構成要素であるプログラム・ヘッダ・テーブルについて説明します。

プログラム・ヘッダ・テーブルは，そのオブジェクト・ファイルに含まれるすべてのセグメントに関する情報（表 A － 2 参照）を持つプログラム・ヘッダ・テーブル・エントリの配列です。

この配列に対するインデックス（添え字）をプログラム・ヘッダ・テーブル・インデックスと呼び，プログラム・ヘッダ・テーブル・エントリの参照は，このプログラム・ヘッダ・テーブル・インデックスを用いて行われます。

表 A － 2 プログラム・ヘッダ・テーブル・エントリの構成要素とその意味

構成要素	意味
type	対応するセグメントのセグメント・タイプ（メモリにロードされるセグメントである場合 LOAD ／補助的な情報を入れたセグメントである場合 NOTE）
offset	対応するセグメントのファイル内オフセット
vaddr	対応するセグメントの仮想アドレス
paddr	対応するセグメントの物理アドレス
filesz	対応するセグメントのファイル上でのサイズ[注]
memsz	対応するセグメントのメモリ上でのサイズ
flags	対応するセグメントのセグメント属性（読み出し可能なセグメントである場合 R ／書き込み可能なセグメントである場合 W ／実行可能なセグメントである場合 X）
align	対応するセグメントの整列条件

注 対応するセグメントに対し NOBITS のセクション・タイプを持つセクション（オブジェクト・ファイル内に実際の値を持たないセクション）を割り付けた場合，memsz と異なる値が設定されます。

A.4 セクション・ヘッダ・テーブル

この節では，オブジェクト・ファイル形式の構成要素であるセクション・ヘッダ・テーブルについて説明します。

セクション・ヘッダ・テーブルは，そのオブジェクト・ファイルに含まれるすべてのセクションに関する情報を持つセクション・ヘッダ・テーブル・エントリの配列です。この配列に対するインデックス（添え字）をセクション・ヘッダ・テーブル・インデックスと呼び，セクション・ヘッダ・テーブル・エントリの参照は，このセクション・ヘッダ・テーブル・インデックスを用いて行われます。

表 A － 3 セクション・ヘッダ・テーブル・エントリの構成要素とその意味

構成要素	意味
name	対応するセクションの名前（セクション名を保持しているストリング・テーブル .shstrtab に対するインデックス）
type	対応するセクションのセクション・タイプ（「A.4.1 セクション・タイプ」参照）
flags	対応するセクションのセクション属性（メモリを占有するセクションである場合 A ／書き込み可能なセクションである場合 W ／実行可能なセクションである場合 X ／グローバル・ポインタ（gp）と 16 ビットのディスプレースメントを用いて参照することのできるメモリの範囲に割り付けるセクションである場合 G）
addr	対応するセクションの先頭アドレス
offset	対応するセクションのファイル内オフセット
size	対応するセクションのサイズ
link	対応するセクションのセクション・ヘッダ・テーブル・インデックス・リンク（「A.4.2 セクション・タイプに依存する要素（link ／ info）」参照）
info	対応するセクションのセクション・タイプに依存する情報（「A.4.2 セクション・タイプに依存する要素（link ／ info）」参照）
addralign	対応するセクションの整列条件
entsize	対応するセクションのエントリのサイズ

A.4.1 セクション・タイプ

次表にセクション・ヘッダ・テーブルの構成要素 type で示されるセクション・タイプとその意味を示します。

表 A － 4 セクション・タイプとその意味

セクション・タイプ	意味
GPTAB	グローバル・ポインタ・テーブル（最初のエントリはコンパイラ，およびアセンブラに対して指定された -G*num* の *num* と 0，2 番目以降のエントリはデータのサイズとワードで整列した場合のサイズ）
NOBITS	オブジェクト・ファイル内に実際の値を持っていないもの（たとえば，初期値の指定されていないデータ）に対するセクション
PROGBITS	オブジェクト・ファイル内に実際の値を持っているもの（たとえば，機械語命令や初期値の指定されているデータ）に対するセクション
REGMODE	レジスタ・モード機能[注]を用いて生成した，リンク可能なオブジェクト・ファイルに存在するセクション（ca850 が内部的に使用したレジスタの本数に関する情報が格納されている）
REL（未サポート）	リロケーション情報
RELA	リロケーション情報
SYMTAB	シンボル・テーブル（A.4.1 参照）
STRTAB	ストリング・テーブル（A.4.2 参照）

注 CA850 のレジスタ・モード指定オプション（-reg）を参照してください。

A.4.2 セクション・タイプに依存する要素（link ／ info）

セクション・タイプ type に依存するセクション・ヘッダ・テーブルの構成要素 link と info の意味を表 A － 5 に示します。

表 A － 5 link と info の意味

セクション・タイプ	link の意味	info の意味
GPTAB	---	対応するデータの割り付けられているセクションのセクション・ヘッダ・テーブル・インデックス
REL（未サポート）	対応するシンボル・テーブルのセクション・ヘッダ・テーブル・インデックス	リロケートされるセクションのセクション・ヘッダ・テーブル・インデックス
RELA	対応するシンボル・テーブルのセクション・ヘッダ・テーブル・インデックス	リロケートされるセクションのセクション・ヘッダ・テーブル・インデックス
SYMTAB	対応するストリング・テーブルのセクション・ヘッダ・テーブル・インデックス	最初に現れるローカルでないシンボルのシンボル・テーブル・インデックス

A.5 セクション

この節では，オブジェクト・ファイル形式の構成要素であるセクションについて説明します。

セクションは，機械語命令，データ，シンボル・テーブル，ストリング・テーブル，デバッグ情報，ライン・ナンバ情報などをその内容として含むオブジェクト・ファイル形式における主要な構成要素です。

セクションは，次に示す条件を満たします。

(1) セクションごとにセクション・ヘッダ・テーブルに対応するセクション・ヘッダ・テーブル・エントリが 1 つ存在します。

(2) セクション・ヘッダ・テーブル・エントリのみが存在しオブジェト・ファイル内に実際の値を持っていないセクションが存在する場合があります（NOBITS のセクション・タイプを持つセクション）。

(3) オブジェクト・ファイル内に実際の値を持っているセクションは，オブジェクト・ファイル内に連続した領域を占めます。

(4) セクションはオブジェクト・ファイル内で領域を共有することはありません。つまり，複数のセクションに属している領域は存在しません。

A.5.1 シンボル・テーブル

この項では，セクションの 1 つの種類であるシンボル・テーブルについて説明します。

シンボル・テーブルは，SYMTAB のセクション・タイプを持つセクションで，そのオブジェクト・ファイルに含まれるすべてのシンボルに関する情報を持つシンボル・テーブル・エントリの配列です。

この配列に対するインデックス（添え字）をシンボル・テーブル・インデックスと呼び，シンボル・テーブル・エントリの参照は，このシンボル・テーブル・インデックスを用いて行われます[注]。

注 シンボル・テーブル・インデックスが 0 のエントリは予約されており，各構成要素は 0 の値を持ちます。

表 A － 6 シンボル・テーブル・エントリの構成要素とその意味

構成要素	意味
name	対応するシンボルの名前（ストリング・テーブル .strtab に対するインデックス）
value	対応するシンボルの値
size	対応するシンボルのサイズ
BIND（info）	対応するシンボルのバインディング・クラス（外部参照の解決において用いられるシンボルである場合 GLOBAL ／外部参照の解決において用いられることのないシンボルである場合 LOCAL）
TYPE（info）	対応するシンボルのタイプ（通常のファイル名である場合 FILE ／関数名である場合 FUNC ／未定義なシンボルである場合 NOTYPE ／通常のラベルを示すシンボルである場合 OBJECT ／セクション名である場合 SECTION ／デバイス・ファイル名である場合 DEVFILE）
other	---
shndx	対応するシンボルに対応するセクションのセクション・ヘッダ・テーブル・インデックス（定数を示すシンボルである場合 ABS ／グローバル・ポインタ（gp）と 32 ビットのディスプレースメントを用いて参照される未定義外部シンボルである場合 COMMON ／グローバル・ポインタ（gp）と 16 ビットのディスプレースメントを用いて参照される未定義外部シンボルである場合 GPCOMMON ／未定義なシンボルである場合 UNDEF の値を取ります）

A.5.2 ストリング・テーブル

この項では，セクションの 1 つの種類であるストリング・テーブルについて説明します。

ストリング・テーブルは，STRTAB のセクション・タイプを持つセクションで，null 文字（\0）が終端である文字列の並びで構成されます。この文字列は，ストリング・テーブルの先頭からのオフセットであるインデックスを用いて参照されます[注]。

オブジェクト・ファイル形式は，シンボルの名前やセクションの名前の保持するためにこれらの文字列を用いており，たとえば，セクション・ヘッダ・テーブル・エントリの構成要素 name は，セクション名を保持しているストリング・テーブル .shstrtab に対するインデックスを持っています。

注 インデックス 0 で表される先頭の 1 バイトは null 文字と定められています。

表 A － 7 ストリング・テーブルにおけるインデックスと文字列の関係

インデックス	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	\0	n	a	m	e	.	\0	V	a	r
+10	i	a	b	l	e	\0	a	b	l	e
+20	\0	\0	x	x	\0					

インデックス	文字列
0	null 文字
+1	name.
+7	Variable
+11	able
+16	able
+24	null 文字

A.5.3 予約セクション

オブジェクト・ファイル形式では，いくつかのセクションが予約セクションとして予約されています。
次表に予約されているセクションの名前とそれらのセクション・タイプ，およびセクション属性を示します。

表 A － 8 予約セクション一覧

名前[注1]	内容	セクション・タイプ	セクション属性
.az_info_j .az_info_r .az_info_ri	AZ850 用情報セクション	PROGBITS	なし
.bss	.bss セクション	NOBITS	AW
.const	.const セクション	PROGBITS	A
.data	.data セクション	PROGBITS	AW
.ext_info .ext_info_boot	フラッシュ／外付け ROM 再リンク機能用情報セクション	PROGBITS	なし
.ext_table	フラッシュ／外付け ROM 再リンク機能用分岐テーブル・セクション	PROGBITS	AX
.ext_tgsym	フラッシュ／外付け ROM 再リンク機能用情報セクション	PROGBITS	なし
.gptabname	グローバル・ポインタ・テーブル[注2]	GPTAB	なし
.pro_epi_runtime	プロローグ／エピローグ・ランタイム呼び出しセクション	PROGBITS	AX
.regmode	レジスタ・モード情報	REGMODE	なし
.relname	リロケーション情報	REL	なし
.relaname	リロケーション情報	RELA	なし
.sbss	.sbss セクション	NOBITS	AWG
.sconst	.sconst セクション	PROGBITS	A
.sdata	.sdata セクション	PROGBITS	AWG
.sebss	.sebss セクション	NOBITS	AW
.sedata	.sedata セクション	PROGBITS	AW
.shstrtab	セクション名を保持しているストリング・テーブル	STRTAB	なし
.sibss	.sibss セクション	NOBITS	AW
.sidata	.sidata セクション	PROGBITS	AW
.strtab	ストリング・テーブル	STRTAB	なし
.symtab	シンボル・テーブル	SYMTAB	なし
.text	.text セクション	PROGBITS	AX
.tibss	.tibss セクション	NOBITS	AW
.tibss.byte	.tibss.byte セクション	NOBITS	AW

表 A － 8 予約セクション一覧

名前[注 1]	内容	セクション・タイプ	セクション属性
.tibss.word	.tibss.word セクション	NOBITS	AW
.tidata	.tidata セクション	PROGBITS	AW
.tidata.byte	.tidata.byte セクション	PROGBITS	AW
.tidata.word	.tidata.word セクション	PROGBITS	AW
.vdbstrtab	デバッグ情報用シンボル・テーブル	STRTAB	なし
.vdebug	デバッグ情報	PROGBITS	なし
.version	バージョン情報セクション	PROGBITS	なし
.vline	ライン・ナンバ情報	PROGBITS	なし

注 1 gptabname，.relname，および .relaname の name の部分は，それぞれそのセクションに対応するセクションの名前を示します。

注 2 リンカにおける -A オプションの処理において用いられる情報です。

付録 B　メッセージ

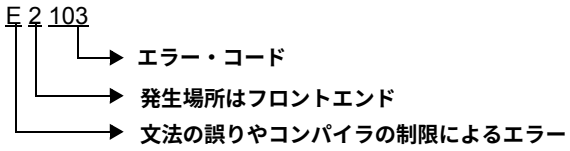
ここでは，CA850 において用いられるメッセージについて説明します。

B.1　出力メッセージ

この節では，CA850 の出力メッセージについて説明します。

B.1.1　メッセージの形式

例



ca850 の出力メッセージには，エラーを発生したモジュール名が付加されます。また，発生したファイル名や行番号が特定される場合は，ファイル名や行番号が付加され，さらに，メッセージ番号が付加されます。メッセージ番号は，生じた誤りのレベルと誤りの発生したモジュールを示しています。

誤りのレベルを示すメッセージ番号の先頭文字の意味は，次のとおりです。

C ... 内部エラー

必ず処理を中止します。

E ... 文法の誤りや制限によるエラー

一定数以上発生した場合，処理を中止します。

例

変数が未定義です。

F ... 致命的エラー

必ず処理を中止します。

例

メモリが足りません。

W ... 警告

処理を続行します。

例

変数に値を設定する前に参照しています。

先頭文字に続く数字は，どのモジュールで生じたエラーであるかを示しています。

1	CA850 ドライバ部（ca850）
2	フロントエンド（cafe）
3	アセンブラ（as850）
4	リンカ（ld850）
5	広域最適化部（opt）
6	コード生成部（cgen）（6000 ～ 6499） 機種依存最適化部（impr850）（6500 ～ 6999）
7	プリオプティマイザ（popt）
8 -	その他

B.1.2 コンパイラ

Cxxxx：内部エラーです（xxxx はエラー番号の番号部分です）。

制限事項になっていないかを確認してください。制限事項になっていない場合は，特約店，または NEC エレクトロニクスの技術サポートまでご連絡ください。

E1305: cannot remove temporary directory '*dir*'

テンポラリ・ファイル作成のために用意した作業用フォルダ *dir* を削除できません。

E1307: cannot unlink temporary file '*file*'

テンポラリ・ファイル *file* を削除できません。

E2043: illegal argument for *function*

関数 *function* の引数が不正です。

E2103: illegal header name

#include で指定されたヘッダ名の文字列が正しくありません。

E2104: cannot find include file '*file*'

#include ディレクティブにおいて指定されたファイル *file* が見つかりません。

E2111: illegal token '*token*'

不正なトークン *token* が認識されました。

E2113: unexpected EOF

構文上許されない位置でファイルが終了しています。

E2114: non-terminated comment

コメントを閉じる “ */ ” がありません。

E2116: illegal expression syntax

前処理制御中の式の記述が正しくありません。

E2117: compiler limit: too long identifier '*name*...' [*num*]

識別子の名前が長すぎます。この処理系の最大値は，内部識別子の場合 1023**，外部識別子の場合** 1022 **です。**

E2118: compiler limit: too many characters in *string* literal [*num*]

文字列リテラルが長すぎます。この処理系の最大値は 32766 **です。**

E2123: compiler limit: too many macro parameters [*num*]

マクロのパラメータの数が多すぎます。この処理系の最大値は 127 **です。**

E2124: illegal macro name '*name*'

マクロ名 *name* **が正しくありません。**

E2125: System reserved macro '*name*' must not be redefined.

マクロ *name* **はシステムで予約されているため再定義できません。**

E2126: System reserved macro '*name*' must not be undefined.

システムで予約されているマクロ *name* **定義を取り消すことはできません。**

E2129: illegal macro parameter '*name*'

マクロのパラメータ *name* **が正しくありません。**

E2130: macro '*name*': mismatch number of parameters

マクロ *name* **のパラメータ数が一致しません。**

E2131: macro '*name*': missing ')'

パラメータを持つマクロ *name* **定義において“)”が欠けています。**

E2133: illegal operand for '#' operator

マクロ定義において“#”演算子にパラメータ以外のものが指定されました。

E2134: compiler limit: too long stringizing result [*num*]

文字列化した結果が長すぎます。この処理系の最大値は 32766 **です。**

E2135: illegal operand for '##' operator

マクロ定義においてトークン連結の記述が正しくありません。

E2136: illegal pasting result

トークン連結を行った結果が正しくありません。

E2137: compiler limit: too long pasting result [*num*]

トークン連結を行った結果が長すぎます。この処理系の最大値は 32766 **です。**

E2138: macro '*name*' illeqal parameter syntax

マクロ *name* **のパラメータの記述が正しくありません。**

E2151: illegal preprocessing directive syntax

前処理指令の記述に誤りがあります。

E2152: illegal number '*name*' in conditional inclusion.

#if に続く式で *name* **のような整数でないものは指定できません。**

E2155: compiler limit: too many conditional inclusion nestings [*num*]

“ #if ” ～ “ #endif ” におけるネスティング回数が限界を越えています。この処理系の最大値は 255 **です。**

E2156: misplaced '#else' or '#elif'

“ #else ” か “ #elif ” の場所が不適当です。

E2157: misplaced '#endif'

“ #endif ” の場所が不適当です。

E2159: illegal include directive syntax

#include **の記述が正しくありません。**

E2164: unexpected non-whitespace before preprocessing directive

前処理指令の前に空白文字類以外の文字があります。

E2165: unrecognized pragma directive '#pragma *directive*', ignored

#pragma *directive* **は認識されません。このプラグマ指令は無視されます。**

E2170: illegal integral/floating constant

整数型，または浮動小数点型定数の表記が正しくありません。

E2171: constant out of range

定数値が表現できる範囲を越えています。

E2173: illegal octal digit

8 **進数の記述に誤りがあります。**

E2174: illegal hexadecimal digit

16 **進数の記述に誤りがあります。**

E2175: octal digit out of range

8 **進数値で表現できる範囲を越えています。**

E2177: empty character constant

文字定数が空です。

E2178: illegal binary digit

2 **進数の記述に誤りがあります。**

E2210: *name*: not defined

name **が定義されていません。**

E2211: redeclaration of *name*

name **が再宣言されています。**

E2213: Nothing is declared.

宣言子が何も指定されていません。

E2214: Void object is not allowed.

void **オブジェクトは許されていません。**

E2217: Undeclared function '*function*' is calld.

宣言のない関数 *function* が呼ばれました。

E2220: Both 'signed' and 'unsigned' are specified.

“ signed **” と “** unsigned **” の両方が指定されています。**

E2221: illegal type specifier combination

型指定子の組み合わせが正しくありません。

E2236: Typedef declaration must not have initializer.

typedef **宣言に初期化子を含むことはできません。**

E2237: too many initializers

初期化子の数が多すぎます。

E2238: illegal initializer

初期化子が正しくありません。

E2240: Local static function is not allowed.

ローカル・スコープで static **関数を宣言することはできません。**

E2250: Array size is not given.

配列のサイズが与えられていません。

E2251: Array size must be greater than zero.

配列のサイズは 0 **より大きくなければなりません。**

E2252: Array type has incomplete element type.

配列の要素の型が不完全です。

E2253: compiler limit : array size is too large [0xffffffff]

配列のサイズが大きすぎます。この処理系の最大値は 0xffffffff **です。**

E2260: compiler limit: complicated type modifiers [*num*]

派生型修飾が多すぎます。この処理系の最大値は 16 **です。**

E2261: illegal storage class specifier combination

記憶クラス指示子の指定の組み合わせが正しくありません。

E2262: illegal use of 'enum'

型指定子 “ enum **” の使用方法が正しくありません。**

E2263: illegal use of 'struct'

型指定子 “ struct **” の使用方法が正しくありません。**

E2265: illegal use of 'union'

型指定子 “ union **” の使用方法が正しくありません。**

E2266: illegal use of 'specifier'

記憶域クラス指定子 “ specifier **” の使用方法が正しくありません。**

E2274: illegal use of 'typedef'

記憶域クラス指定子 “ typedef ” の使用方法が正しくありません。

E2280: Void function cannot return value.

void 型関数で戻り値が指定されています。

E2281: Function has illegal storage class.

関数に対する記憶クラスの指定が正しくありません。

E2282: Array of function is not allowed.

関数の配列は許されていません。

E2283: illegal return type: function

関数の戻り値を関数型とすることはできません。

E2284: illegal return type: array

関数の戻り値を配列型とすることはできません。

E2285: 'Void' in parameter list

関数宣言で引数宣言並びの中に void 型が指定されています。void 型は単一でのみ使用できます。

E2286: Function requires return value.

戻り値を持つ関数において戻り値が指定されていません。

E2288: return type mismatch *type1* (*type2*)

return 文で示された戻り値の型 *type2* が関数の戻り型 *type1* と一致しません。

E2290: argument type mismatch *type1* (*type2*)

実引数の型 *type2* が関数宣言時の仮引数の型 *type1* と一致しません。

E2292: Argument *name* is missing.

関数定義において宣言されている仮引数名 *name* が見つかりません。

E2296: illegal first argument '...', requires a named argument

関数の第一引数には “ ... ” は使えません。

E2300: 'Struct'/'union' size must not be zero.

構造体，または共用体のサイズは 0 にはできません。

E2301: illegal bit-field type

ビット・フィールドに対し指定することのできない型が指定されています。

E2303: illegal bit-field size

ビット・フィールドの幅を指定する定数式の値は，指定された型のオブジェクトを構成するビット数を越えることはできません。

E2304: '*name*' has incomplete type.

***name* の型が不完全型です。**

E2305: Field '*name*' declared as a function.

メンバ *name* の型が関数型です。

E2347: Local extern '*symbol*' is put into the next unit.

局所スコープで外部リンケージを持つ宣言子 *symbol* が初期化されています。

E2349: Initialization of non-auto pointer using non-number initializer is not position independent.

自動変数以外のポインタ変数の値以外の初期値を用いた初期化の指示に対するコードは，ポジション・インディペンデントではありません。

E2361: New style function definition has old style parameter declaration(s).

新しい関数定義形式で，古い引数宣言形式が使われています。

E2374: The bit-field object '*name*' is put into the next unit.

ビット・フィールド *name* は境界を越えるため，次の領域に配置されます。

E2401: syntax error

構文に誤りがあります。

E2402: Label '*label*' not defined

ラベル *label* が定義されていません。

E2411: label is not in switch

case **ラベル，または** default **ラベルが** switch **文の中にありません。**

E2412: duplicate'case *num*:' in switch

switch **文中の** case **ラベル *num* が重複しています。*num* は数値として展開されている場合があります。**

E2413: duplicate 'default:' in switch

switch **文中で** default **ラベルが重複しています。**

E2414: 'break' not in loop nor switch

“ break **” が繰り返し文，または** switch **文の外にあります。**

E2415: 'continue' not in loop

“ continue **” が繰り返し文の外にあります。**

E2420: argument *num* expected for function call *function*

関数呼び出し *function* において *num* 番目以降の引数が指定されていません。

E2421: unexpected argument *num* for function call *function*

関数呼び出し *function* において *num* 番目以降の引数が余分に指定されています。

E2422: undefined static function '*function*'

呼び出された static **関数 *function* がファイル内で定義されていません。**

E2510: cannot cast: *type1* to *type2*

type1 **から** *type2* **へのキャストはできません。**

E2511: *expression* must be arithmetic or pointer type.

expression は算術型，またはポインタ型にしてください。

E2512: *expression* must be arithmetic type.

expression は算術型にしてください。

E2513: *expression* must be pointer type or zero.

expression はポインタ型，または 0 にしてください。

E2515: *expression* must be integral type.

expression は整数型にしてください。

E2516: *expression* must be constant expression.

expression は定数式にしてください。

E2517: One of the operands for '[ ]' must be pointer type and the other must be of integral type.

“[]”に対するオペランドの一方はポインタ型に，もう一方は整数型にしてください。

E2518: illegal operand for unary '&'

単項演算子“&”のオペランドが正しくありません。

E2519: *exception* has occurred at compile time.

コンパイル時に浮動小数点関連で例外 *exception* が発生しました。

E2522: *name* is not a member.

name は集成体のメンバではありません。

E2523: illegal LHS of '*operator*' operator(must be modifiable Lvalue)

operator 演算子の左辺に代入先として不正なものが置かれています。

E2524: illegal type combination for '*operator*'(*type1*, *type2*)

operator 演算子に対する型の組み合わせ（*type1*，*type2*）が正しくありません。

E2526: Operands of '*operator*' operator must have same type(*type1*, *type2*).

operator 演算子の両辺（*type1*，*type2*）の型は同じ型にしてください。

E2529: invaid use of void expression

void 式が正しくありません。

E2530: Operand of '()' must be function type.

“()”演算子のオペランドは関数にしてください。

E2532: Operand of '*operator*' must be pointer type.

operator 演算子のオペランドはポインタ型にしてください。

E2533: Operand of '.' must be 'struct'/'union' object.

“.”演算子は構造体，または共用体で使用してください。

E2535: Operand of '-' must be pointer to 'struct'/'union' object.

“ - ” 演算子は構造体，または共用体へのポインタで使用してください。

E2550: Operand of 'sizeof' must not be *type*.

“ sizeof **” のオペランドに** *type* **を指定できません。**

E2551: unknown size('struct', 'union' or array)

オブジェクト・サイズが必要な演算子にサイズの不明な集成体が指定されました。

E2552: unknown size (function)

オブジェクト・サイズが必要な演算子にサイズの不明な関数が指定されました。

E2553: cannot convert non-Lvalue array to pointer

左辺値でない配列はポインタに変換できません。

E2556: unknown size ('enum')

オブジェクトのサイズが必要な演算子にサイズの不明な列挙子が指定されました。

E2630: unrecognized interrupt request name '*name*'

#pragma **指令により不正な割り込み要求が指定されています。**

E2631: Interrupt request name '*name*' is already specified.

割り込み要求名 *name* **はすでに指定されています。**

E2632: illegal directive '#pragma *directive*', function name must be specified

#pragma *directive* **指令には関数名が必要です。**

E2633: cannot specify interrupt attribute 'direct', function '*function*' is already defined.

関数定義よりも後ろで，割り込みハンドラを直接配置指定することはできません。

E2636: Multiple interrupt request names are specified for function '*function*', 'direct' cannot be specified.

関数 *function* **には，複数の割り込み要求が指定されています。複数の割り込みが指定された関数には，直接配置指定（**direct**）ができません。**

E2638: Interrupt function must be void type.

割り込みとして宣言された関数の戻り型は void **型にしてください。**

E2639: illegal function type for software interrupt function, must be void (unsigned int).

ソフトウエア例外の割り込み（トラップ割り込み）として宣言された関数は引数に unsigned int **型を** 1 **つしか持つことができません。**

E2640: illegal function type for interrupt function, must be void (void).

割り込みとして宣言された関数（ソフトウエア例外を除く）は引数を持つことはできません。

E2641: cannot call interrupt function

割り込みとして宣言された関数を呼び出すことはできません。

E2642: Function '*function*' is already defined, 'block_interrupt' must be specified before function definition.

割り込み禁止は関数定義の後ろでは指定できません。

E2644: Function '*function*' is already defined without '__interrupt'.

関数 *function* が割り込みハンドラと指定されましたが，*function* はすでに割り込み指定なしで定義されています。

E2646: Both interrupt and RTOS interrupt attributes are specified.

通常の割り込みとリアルタイム OS 用の割り込みを同時に指定することはできません。

E2647: Specifying interrupt name '*name*' is not allowed.

割り込み要求名に RESET，および RST は指定できません。

E2648: unknown cpu type, cannot use interrupt request name

デバイス指定がないため，割り込み要求名を使用することはできません。

E2649: Interrupt function '*function*' with 'direct' is undefined.

直接配置指定された関数 *function* が，ファイル内に定義されていません。

E2650: illegal directive ' #pragma section ', section name must be specified

#pragma 指令によるセクション割り当てにおいて，セクション名が指定されていません。

E2651: illegal directive ' #pragma section ', unrecognized section name '*name*'

#pragma 指令によるセクション割り当てにおいて，不正なセクション名 *name* が指定されています。

E2652: illegal directive ' #pragma section ', ' begin ' or ' end ' must be specified

#pragma 指令によるセクション割り当てには “ begin ” や “ end ” が必要です。

E2653: Directive ' #pragma section ' is nested.

#pragma 指令によるセクション割り当て指定がネストしています。

E2654: inconsistent section for '*symbol*'

シンボル *symbol* に対してセクションが矛盾しています。

E2655: misplaced ' #pragma section section end '

“ #pragma section section end ” の位置が不適当です。

E2660: cannot write, read only I/O register '*regname*'

リード属性のみを持つ内部周辺 I/O レジスタ *regname* にデータを書き込むことはできません。

E2661: cannot read, write only I/O register '*regname*'

ライト属性のみを持つ内部周辺 I/O レジスタ *regname* からデータを読み出すことはできません。

E2662: cannot access for I/O register bit number '*regname*'

内部周辺 I/O レジスタ *regname* に対するビット・アクセスの記述において，アクセス不可能な位置を指定しています。

E2663: I/O register bit number must be integral type.

部周辺機能用レジスタに対するビット位置の指定は整数値で記述してください。

E2664: Specifying bit number for I/O register '*regname*' is not allowed.

内部周辺 I/O レジスタ *regname* のビットに対して，ビット・アクセスを指定することはできません。

E2665: unknown cpu type, cannot use I/O register

ターゲット・デバイスが不明のため，内部周辺 I/O レジスタを使用することはできません。

E2666: illegal operand (I/O register '*regname*') for unary '&'

内部周辺 I/O レジスタ *regname* のアドレスを求めることはできません。

E2670: unexpected EOF, missing '#pragma endasm'

アセンブラ挿入の終わりの指定が見つかりません。

E2681: First argument for __set_il is out of range.

割り込みレベルの値が指定可能な範囲を越えています。割り込みレベルとして設定できるのは -1 ～ +8 の整数です。

E2682: Second argument for __set_il must be string literal ("Interrupt Request Name")

割り込みレベルを設定する関数の第二引数には，割り込み要求名を示す文字列を指定してください。

E2685: illegal argument for __set_il(int, "Interrupt Request Name")

割り込みレベルを設定する関数の引数が間違っています。第一引数は整数型，第二引数は割り込み要求名を指定してください。

E2692: Both interrupt attribute and 'rtos_task' are specified.

関数に対して，リアルタイム OS 用のタスクと割り込みを同時に指定することはできません。

E2693: Function '*function*' is already defined, 'rtos_task' must be specified before function definition.

関数を，関数定義よりも後ろでリアルタイム OS 用のタスクとして指定することはできません。

E2694: Function '*function*' is already defined without '__rtos_interrupt'.

関数 *function* が割り込みハンドラと指定されましたが，*function* はすでにリアルタイム OS 用の割り込み指定なしで定義されています。

E2695: cannot call rtos_task function

リアルタイム OS 用のタスクとして指定された関数を呼び出すことはできません。

E2696: Rtos system call '*function*' is already defined, cannot specify '#pragma kind'

すでに関数 *function* と同名の関数が定義，または宣言されています。リアルタイム OS の #pragma 指令に

よりシステム・コールを有効にすることができません。

E2697: Rtos system call '*name*' is called in the function, for which rtos interrupt attribute is not specified.

リアルタイム OS 用の割り込み指定のない関数で，システム・コール *name* が呼ばれています。*name* を通常の関数呼び出しとみなします。

E2698: cannot call rtos_interrupt function

リアルタイム OS 用の割り込みハンドラとして指定された関数を呼び出すことはできません。

E2701: Duplicated GP symbol for RTOS interrupt function '*function*'

リアルタイム OS 用の割り込みハンドラとして指定された関数 *function* には，すでに別の gp シンボルが割り当てられています。

E2702: Specifying interrupt name '*name*' is not allowed for rtos_interrupt.

割り込み要求名に *name* は指定できません。

E2712: unexpected end-of-line (missing '] ')

セクション・ファイルにおけるセクション名は“[]”で囲んでください。

E2713: unexpected character(s) '*token*'

セクション・ファイルに余分なトークン *token* があります。セクション・ファイルの形式を確認してください。

E2714: Variable, function or file name is missing.

セクション・ファイルの変数情報の記述が不正です。

E2715: illegal function/variable name '*symbol* '

シンボル *symbol* は，不正な関数名，または変数名です。

E2716: Section name is not specified.

セクション名が指定されていません。

E2717: unrecognized section name '*section*'

不正なセクション名 *section* が指定されています。

E2746: Too long section name[256]

セクション名が長すぎます。256 文字以下にしてください。

E2747: inconsistent section name for '*symbol*'

シンボル *symbol* に対してセクション名が矛盾しています。

E2749: Function '*function*' is already defined without '__multi_interrupt'.

関数 *function* が多重割り込み関数と指定されましたが，すでに多重割り込み指定なしに定義されています。

E2750: Both interrupt and multi interrupt attributes are specified.

通常の割り込みと多重割り込みを，同時に指定することはできません。

E2751: Both RTOS interrupt and multi interrupt attributes are specified.

リアルタイム OS 用の割り込みと多重割り込みを，同時に指定することはできません。

E2752: cannot call *function* function

***function* 関数を呼ぶことはできません。**

E2760: unknown cpu type, cannot use *directive*

ターゲット・デバイス指定がないため，*directive* を使用することはできません。

E2781: result of comparison is always false

比較式が常に偽になります。

E2783: statement with no effect

意味のない文があります。

E2785: Conversion may lose significant digit

データが失われている可能性があります。

E7201: multiple defined symbol '*symbol*'

***symbol* 多重定義シンボル *symbol* が存在します。**

E7202: redeclaration of '*symbol*'

***symbol* が再宣言されています。**

E7203: undefined symbol '*symbol*'

***symbol* が未定義です。**

E7204: undefined label (.L*num*)

.L*num* のラベルが未定義です。

E7205: Argument type mismatch is detected where '*caller*' calls '*callee*'.

インライン展開時に *caller* と *callee* の引数の型に違いがあります。

E7206: Return value type mismatch is detected where '*caller*' calls '*callee*'.

インライン展開時に *caller* と *callee* の戻り値の型に違いがあります。

E7207: interrupt request '*name*' already specified

割り込み要求名 *name* はすでに指定されています。

E7208: inconsistent section for '*symbol*'

***symbol* に対してセクションが矛盾しています。**

F1001: out of memory

メモリが足りません。

F1002: cannot recover from previous errors

これ以前に発生したエラーのために，処理を継続できません。

F1102: invalid argument of option '*option*'

オプション *option* の引数が不正です。

F1103: nested command file '*file*'

コマンド・ファイル *file* がネストしています。

ネストはできません。

F1104: Argument of -reg option requires 22, 26 or 32.

-reg **オプションの引数には** 22，26，32 **のいずれかを指定します。**

F1105: cannot use '*option1*' option with '*option2*' option

オプション *option1* とオプション *option2* は同時指定できません。

F1106: cannot specify output file name of -o with many source files.

複数のソース・ファイルを入力する場合，オプション -o **の出力ファイル名は指定できません。**

F1107: Register '*rnum*' is reserved for compiler system.

レジスタ r*num* **はコンパイラ・システムで予約されています。**

F1202: *module*: not found

起動すべきモジュール *module* が見つかりません。

F1203: *module*: exec failed

モジュール *module* の実行に失敗しました。

F1292: too long argument

モジュール実行時に引数が 128 **バイトを越えました。**

引数をコマンド・ファイルにしてください。

F1302: '*file*': illegal output file name

-o **オプションで指定した出力ファイル名 *file* は，入力ファイル名と同じにはできません。*file* を変更して指定してください。**

F1303: cannot open file '*file*'

ファイル *file* をオープンできません。

F1304: cannot create temporary directory

テンポラリ・ファイルを作成するための作業用フォルダが作成できません。

F1306: cannot open temporary file '*file*'

テンポラリ・ファイル *file* をオープンできません。

F1309: ' *file* ': illegal output file name of *option*

オプション *option* で指定された出力ファイル名 *file* は，入力ファイル名と同じにはできません。*file* を変更してください。

F1310: cannot create directory '*dir*'

フォルダ *dir* を作成できません。

F1311: cannot find device file

デバイス・ファイルが見つかりません。

F2001: illegal command path

指定されたコマンドのパスが正しくありません。

F2002: compiler limit: too long command path [*num*]

コンパイラ制限 : 指定されたパスの長さが限界を越えています。この処理系の最大値は 1024 **です。**

F2003: out of memory

メモリが足りません。

F2004: too many errors

エラーの発生回数が定められている回数を越えたため，コンパイルを中止しました。

F2005: cannot open output file '*file*'

出力ファイル *file* をオープンできません。

F2006: cannot open input file '*file*'

入力ファイル *file* をオープンできません。

F2007: cannot write file '*file*' (errno=*num*)

ファイル *file* 書き込み中にエラー番号 *num* のエラーが発生しました。

F2010: illegal option '*option*'

オプション *option* の指定が正しくありません。

F2011: compiler limit: too many *option* options [*num*]

オプション *option* の指定回数が限界を越えています。この処理系の最大値は *num* です。

F2014: Both '*option1*' and '*option2*' cannot be specified.

オプション *option1* と *option2* は同時に指定できません。

F2020: compiler limit: scope level too deep [*num*]

スコープ・レベルの深さが限界を越えています。この処理系の最大値は 127 **です。**

F2040: compiler limit: too many parameters [*num*]

関数の仮引数の数が多すぎます。この処理系の最大値は 255 **です。**

F2105: compiler limit: too long file name '*file*' [*num*]

ファイルの名前 *file* が長すぎます。この処理系の最大値は 1024 **です。**

F2106: Non empty file must end in new-line character.

空でないファイルは改行文字で終了してください。

F2110: unknown character '*character*'

不正な文字 *character* が用いられています。

F2112: compiler limit: too many characters in logical source line [*num*]

論理ソース行の文字数が限界を越えています。この処理系の最大値は 32768 **です。**

F2115: non-terminated *string* literal

文字列を閉じる“"”がありません。

F2119: compiler limit: string buffer overflow [*num*]

コンパイラ制限 : **文字列バッファが溢れました。この処理系の最大値は** 32768 **です。**

F2120: compiler limit: preprocessor token buffer overflow [*num*]

マクロ定義において，展開文字列の長さが限界を越えています（*num* 文字分のバッファが足りません）。

[備考]

マクロ定義において，展開文字列の長さが限界を越えていた場合に出力されます。マクロ定義数の限界を上げるオプション -Xm*num*（*num* **は，**-Xm **を省略した場合は** 2,047**，最大** 32,767 **まで指定可能）で，限界値を変更して見てください。**

なお，このオプションはプリプロセッサで使用するバッファのサイズを大きくするものであり，これによって何文字分のバッファが確保されるのかという具体的な数値が出せません。

F2121: compiler limit: too many macro definitions [*num*]

マクロ定義の数が限界を越えています。この処理系の最大値は *num* です。

F2122: compiler limit: too long macro name '*name*' [*num*]

マクロ名 *name* が長すぎます。この処理系の最大値は 1023 **です。**

F2128: redeclared macro parameter '*name*'

マクロのパラメータ *name* が再定義されています。

F2153: unexpected non-whitespace before preprocessing directive

前処理指令の前に空白文字類以外の文字があります。

F2154: undefined control

“#”に続く前処理指令の記述が正しくありません。

F2158: compiler limit: too many include nestings [*num*]

#include **ディレクティブにおけるネスティングの回数が限界を越えています。この処理系の最大値は** 50 **です。**

F2160: *errmsg*

errmsg で示されるエラーが発生しました。ソース・プログラム中で #error ディレクティブが使用されると，このメッセージが出力されます。

F2230: compiler limit: initialization nests too deep [*num*]

初期化子リストのネストが深すぎます。この処理系の最大値は 100 です。

F2410: compiler limit: too many case labels [*num*]

switch 文中の case ラベルの個数が限界を越えています。この処理系の最大値は 1025 です。

F2608: cannot recover from earlier errors

先に発生したエラーのために，処理を継続できません。

F2620: unknown cpu type, cannot compile

ターゲット・デバイス指定がないため，コンパイルできません。

F2622: duplicated cpu type

オプション，または #pragma 指令でターゲット・デバイス指定が重複しています。

F2623: cannot find device file

指定されたターゲット・デバイスに相当するデバイス・ファイルがありません，またはターゲット・デバイスの指定が間違っています。

F2624: device file read error

デバイス・ファイルの読み込みに失敗しました。デバイス・ファイルが壊れている可能性があります。

F2625: illegal placement '#pragma cpu'

デバイス名を指定する #pragma 指令の位置が不正です。デバイス指定は C 言語の構文より前に記述してください。

F2626: illegal cpu type: *type*

デバイス指定の対応がとれていません。ca850 に対応するデバイス指定を行ってください。

F2628: device file version mismatch, cannot use version '*version*'

デバイス・ファイルのフォーマット・バージョンが不正です。

F5001: unknown option '*option*'

不正なオプション *option* が指定されました。

F5005: invalid argument of option '*option*'

オプション *option* の引数が不正です。

F5104: out of memory

メモリが足りません。

F5106: exception *exception* has occurred at compile time.

コンパイル時に浮動小数点関連で例外 *exception* が発生しました。

F5601: cannot allocate register to '*symbol*'

変数 *symbol* にレジスタを割り付けることはできません。

F5901: cannot open file '*file*'

ファイル *file* をオープンできません。

F5902: cannot write file '*file*'(errno=*num*)

ファイル *file* 書き込み中にエラー番号 *num* のエラーが発生しました。

F6000: cannot open file '*file*'

ファイル *file* をオープンできません。

F6002: cannot unlink file '*file*'

ファイル *file* を削除できません。

F6003: -wreg num is out of range (*num1*=<num=<*num2*).

-wreg **オプションに指定された** num **の値が範囲外です。*num1* 以上 *num2* 以下の値にしてください。**

F6004: -rreg num is out of range(*num1*=<num=<*num2*).

-rreg **オプションに指定された** num **の値が範囲外です。*num1* 以上 *num2* 以下の値にしてください。**

F6005: cannot write file '*file*'(errno=*num*)

ファイル *file* 書き込み中にエラー番号 *num* のエラーが発生しました。

F6203: out of memory

メモリが足りません。

F6500: unknown option '*option*'

不正なオプション *option* が指定されました。

F6510: too many files

ファイル名の指定が多すぎます。

F6520: out of memory

メモリが足りません。

F6530: cannot open file '*file*'

ファイル *file* をオープンできません。

F6540: cannot write file '*file*'

ファイル *file* 書き込み中にエラーが発生しました。

F6550: cannot read file '*file*'

ファイル *file* 読み込み中にエラーが発生しました。

F6560: cannot create file '*file*'

ファイル *file* を生成できません。

F6580: input line is too long

入力ファイルの一行の長さが長すぎます。

F7000: too many errors

エラーの発生回数が定められている回数を越えたため，コンパイルを中止しました。

F7001: unknown option '*option*'

不正なオプション *option* が指定されました。

F7002: invalid argument of option '*option*'

オプション *option* の引数が不正です。

F7003: nested command file '*file*'

コマンド・ファイル *file* がネストしています。ネストはできません。

F7004: no input file

入力ファイルの指定がありません。

F7005: cannot open file '*file*'

ファイル *file* をオープンできません。

F7006: archive symbol table and archive member mismatch

アーカイブ・シンボル・テーブルに異常があります。

F7007: unknown file type '*file*'

***file* のファイル・タイプがわかりません。**

F7009: out of memory

メモリが足りません。

F7010: multiple defined symbol '*symbol*'

多重定義シンボル *symbol* が存在します。

F7011: duplicated cpu type

オプション，または #pragma **指令でターゲット・デバイス指定が重複しています。**

F7012: cannot write file '*file*'(errno=*num*)

ファイル *file* 書き込み中にエラー番号 *num* のエラーが発生しました。

W1111: sorry, not implemented option '*option*', ignored

オプション *option* はサポートしていません。無視されます。

W1112: -G option needs size(>=0): ignored

-G **オプションには，続けてサイズの指定が必要です。∞（無限大）が指定されたものとみなしました。**

W1114: file '*file*' with unknown suffix passed to ld

ファイル *file* は不明な拡張子のファイルです。ld850 **へ渡します。**

W1116: sorry, '*suffix*' file not supported, ignored

拡張子が *suffix* のファイルはサポートしていません。無視されます。

W1119: *option1* option overrides *option2* option.

オプション *option1* を指定したことによりオプション *option2* が無効になりました。

W1120: *option1* option obsolete, use *option2* instead

オプション *option1* は旧式のオプションです。オプション *option2* を使用してください。

W1123: '*option1*' option ignored, for '*option2*' option

オプション *option2* が指定されたのでオプション *option1* は無視されます。

W1126: -cn option must be used with V850 core, used -cnv850e option instead

オプション -cn **は** V850 **コアのデバイスで使用しなければなりません。代わりにオプション** -cnv850e **が使用されます。**

W1127: '*option*' option is not supported for V850 core.

オプション *option* は V850 **コアのデバイスではサポートしていません。**

W1128: cannot find programmable peripheral I/O registers, ignored

プログラマブル周辺 I/O **レジスタが見つかりません。オプションは無視されます。**

W1129: -cn option must be used with V850 core, used -cnv850e2 option instead

オプション -cn **は，**V850 **コアのデバイスで使用しなければなりません。代わりにオプション** -cnv850e2 **が使用されます。**

W1130: -cnv850e option must be used with V850E core, used -cnv850e2 option instead

オプション -cnv850e **は，**V850Ex **コアのデバイスで使用しなければなりません。代わりにオプション** -cnv850e2 **が使用されます。**

W1308: output file of *option1* overrides output file of *option2*

オプション *option2* の出力ファイルを指定したことにより，オプション *option1* の出力ファイルが上書きされます。

W2015: illegal warning message number '*num*' specified by '*option*'

option **で指定された** *num* **は正しい警告メッセージ番号ではありません。**

W2042: illegal argument for _rcopy

コピー・ルーチン _rcopy **の引数が不正です。**

W2107: Non empty file is expected to end in new-line character.

空ではないファイルは，改行文字で終了してください。

W2127: redefined macro name '*name*'

マクロ名 *name* が再定義されています。あとから定義されたものが有効になります。

W2132: macro recursion '*name*'. Macro is expanded only one time.

マクロ再帰があります。マクロは一度しか展開されません。

W2150: unexpected character(s) following directive '*directive*'

前処理指令 *directive* の後ろに余分なトークンがあります。余分なトークンは無視されます。

W2161: unexpected non-whitespace before preprocessing directive

前処理指令の前に空白文字類以外の文字があります。

W2162: unrecognized pragma directive '#pragma *directive*', ignored

#pragma *directive* は認識されません。このプラグマ指令は無視されます。

W2163: Digit sequence after '#line' is interpreted as a decimal interger.

#line に続く数字列は 10 進数として解釈されます。

W2166: recognized pragma directive '#pragma *directive*'

前処理指令は #pragma *directive* と認識されます。

W2172: constant out of range

定数値が表現できる範囲を越えています。下位の有効な桁数分だけが指定されたものとみなします。

[備考]

定数値が型に対して表現できる範囲を越えた場合に出力されます。

例

char 型変数に対して 0xff 以上の値の代入

W2176: hexadecimal digit out of range

16 進数値で表現できる範囲を越えています。この処理系では最後の 2 文字を有効とします。

W2180: cannot convert *code1* code into *code2* code (data *data1 data2 data3*)

code1 から *code2* へのコード変換ができません。

code1: ホスト環境の文字コード
code2: 実行環境の文字コード
*data**: 変換できなかったデータ（16 進表記）

W2212: Declaration of *name* hides parameter.

引数と同名のシンボル *name* の宣言があるために，シンボルを有効とし，引数の宣言が隠されます。この警告メッセージは -ansi オプション指定時，E2211 のエラー・メッセージとなります。

W2215: Undeclared function '*function*' is called.

宣言のない関数 *function* が呼ばれました。このメッセージは，-w2 指定時のみ出力されます。

W2216: Nothing is declared.

宣言子が何も指定されていません。

W2222: Plain int bitfield is treated as unsigned int.

単なる int のビット・フィールドを unsinged int とみなします。

W2231: Initialization of non-auto pointer using non-number initializer is not position independent.

自動変数ではないポインタ変数の値以外の初期値を用いた初期化の指示に対するコードは，ポジション・インディペンデントではありません。

このメッセージは，-w2 指定時のみ出力されます。

[備考]

自動変数ではないポインタ変数値以外の値を初期値として用い，初期化指示に対するコードが存在する場合に出力されます。ポジション・インディペンデント・コード（PIC）用にサポートされた警告メッセージです。

例

```
int *ip = &i;
```

上記の初期化は，ip **が自動変数ならば，実行時に値が代入されるので** PIC **となります。**ip **が自動変数でなければ，コンパイル時に値が決まり，**PIC **にはならないので，この警告メッセージを出力します。**

CA850 **で，プログラムを** TEXT **セグメント，変数を** DATA **セグメントに配置した場合，プログラムの分岐や，変数の参照は** TP **レジスタ，**GP **レジスタ相対のコードになります。これは，実行時，リンク時に配置したアドレスとは異なるアドレスにコピーされたとしても，**TP **レジスタ，**GP **レジスタの値を設定し直せば，正常に動作するプログラムになります。**

PIC **とは，配置されたアドレスに依存しないコードのことになります。そのため，外部変数であるポインタ変数にアドレスを初期値として代入する場合には，リンク時のアドレスが代入されてしまうために，実行時にアドレスが変更されると，正常に動作しません。つまり，**PIC **にはならないので，この警告メッセージを出力するような仕様になっています。**

ただし，r0 **相対（**CONST **セグメントなど）が使用されている場合には，アドレスが絶対番地（**0 **番地からの相対のため）になるため，**PIC **にはなりません。**

W2244: 'asm' used out of function is not supported completely.

関数の外で記述されたアセンブラ記述asm**,および関数の外で記述された**#pragma asm〜#pragma endasm **の間のアセンブラ記述には制限があります。**

W2254: zero sized array '*symbol*'

配列のサイズに 0 **が指定されてます。**

W2267: illegal use of '*specifier*'

記憶域クラス指定子“ *specifier* **”の使用方法が正しくありません。**

W2287: Function requires return value.

戻り値を持つ関数において，関数の戻り値が指定されていません。戻り値に 0 **が指定されたものとみなします。**

W2289: return type mismatch *type1* (*type2*)

return **文で示された戻り値の型** *type2* **が，関数の戻り型** *type1* **と一致しません。**

W2291: argument type mismatch *type1* (*type2*)

実引数の型 *type2* **が，関数宣言時の仮引数の型** *type1* **と一致しません。**

W2293: Type specifier of argument *name* is missing.

関数定義において，宣言されている仮引数名 *name* **の型指定子が省略されています。**int **型とみなします。この警告メッセージは** -ansi **オプション指定時，**E2292 **のエラー・メッセージとなります。**

W2302: illegal bit-field type

ビット・フィールドに対し，ANSI **仕様で指定することのできない型が指定されています。指定された型の整列条件でパディングします。**

この警告メッセージは -ansi **オプション指定時，**E2301 **のエラー・メッセージとなります。このメッセージは，**-w2 **指定時のみ出力されます。**

W2306: The bit-field object '*name*' is put into the next unit.

ビット・フィールド *name* は境界を越えるため，次の領域に配置されます。このメッセージは，-w2 **指定時のみ出力されます。**

W2373: used '&' for member of packed structure

パッキングされた構造体のメンバに対しアドレスを使用しています。

[備考]

構造パッキングを行って，次のいずれかの条件に該当する場合，データのアクセスは，デバイスのデータ・アライメントに従いアドレスをマスクして行われるため，構造体メンバのアドレスでのアクセスで，データの消失や切り捨てが生じます。

- **ミス・アライン・アクセスに対応していないデバイスを使用している**
- **ミス・アライン・アクセスに対応したデバイスで，ミス・アライン・アクセスを禁止している**

例

```
struct test {
   char c;     /* offset 0 */
   int  i;     /* offset 1-4 */
} test;
int *ip, i;
void func(){
   i = *ip;    /* マスクされたアドレスでアクセスされる */
}
void func2(){
   ip = &(test.i);
}
```

W2380: function returns address of local variable

関数の戻り値に自動変数のアドレスが指定されています。自動変数のアドレスの返却をしないでください。次の例のように，自動変数のアドレスを返却値にしないようにしてください。

例

```
void* func(void)
{
      int i;
      return &i;
}
```

W2416: over 0x2000 tables, ignored -Xcase=table option

テーブル数が 0x2000 **を越えましたので** if-else **形式で出力します。**-Xcase=table **は無視されます。**

W2520: Immediate for shift operator is out of range.

シフト命令に対し指定されたイミーディエトの値が指定可能な値の範囲を越えています。下位の有効な桁数分だけが指定されたものとみなします。

W2521: division by zero

コンパイル時に行われる定数式の演算において，0 **による除算が生じました。定数式に** 0 **が指定されたものとみなします。**

W2525: illegal type combination for '*operator*' (*type1*, *type2*)

operator **演算子に対する型の組み合わせ（***type1***，***type2***）が正しくありません。型変換して処理を続行します。この警告メッセージは** -ansi **オプション指定時，**E2524 **のエラー・メッセージとなることがあります。**

W2527: Operands of '*operator*' operator must have same type (*type1*, *type2*).

operator 演算子の両辺の型は同じ型でなければなりません（*type1*，*type2*）。

W2554: cannot convert non-Lvalue array to pointer

左辺値でない配列は，ポインタに変換できません。この警告メッセージは -ansi オプション指定時，E2553 のエラー・メッセージとなります。

W2555: expression *expression* must have enumeration type.

expression は enum 型にしてください。

W2600: ignored option '*option*'

オプション *option* は，無視されます。

W2601: *category* is not supported now.

category で示された機能は現在サポートされていません。

W2606: Wide-character is not supported.

ワイド文字はサポートされていません。ワイド文字は無視されます。

W2607: Multibyte-character is not supported.

マルチバイト文字はサポートされていません。マルチバイト文字は無視されます。

W2609: Specified warning message number '*num*' is not supported. Warning message number W2000-W2999 is supported now.

指定された警告メッセージ番号 *num* はサポートされておりません。対応している警告メッセージ番号は 2000 番台です。

W2621: duplicated cpu type, command line option is used

ターゲット・デバイスの指定が重複しています。プロジェクト・ファイル設定時，または -cpu オプションで指定されたターゲット・デバイスが有効となります。

W2634: Interrupt attribute is specified for function '*function*', previously specified 'block_interrupt' is ignored.

割り込み禁止指定されている関数 *function* が，割り込みハンドラとして指定されました。割り込みハンドラは割り込み禁止として扱われるため，無駄な割り込み禁止指定は無視されます。

W2635: Interrupt attribute is already specified for function '*function*', 'block_interrupt' is ignored.

関数 *function* は，すでに割り込みハンドラとして宣言されています。割り込みハンドラは割り込み禁止として扱われるため，無駄な割り込み禁止指定は無視されます。

W2637: Interrupt function cannot be inlined, 'inline' is ignored.

割り込みとして宣言された関数には，“ inline ” を指定することはできません。inline 指定は無視されます。

W2643: Interrupt attribute is specified for function '*function*', previously specified 'inline' is ignored.

インライン指定されている関数 *function* が割り込みハンドラとして指定されました。inline 指定は無視されます。

W2656: unknown size, cannot specified const/sconst section

サイズが不明な変数に対して const**，または** sconst **セクション指定をすることはできません。**

W2671: Function '*function*' is already defined, directive '#pragma inline' is ignored.

インライン指定は関数定義の前に記述しなければなりません。inline **指定は無視されます。**

W2683: Second argument '*name*' for __set_il is not interrupt request name.

割り込みレベルを設定する関数の第二引数に指定された *name* **は，割り込み要求名ではありません。割り込みレベルは設定されません。**

W2684: cannot set interrupt level for '*name*'

割り込み要求名 *name* **には，割り込みレベルを設定することはできません。割り込みレベルは設定されません。**

W2690: 'Rtos_task' is specified for function '*function*', previously specified 'inline' is ignored.

インライン指定されている関数が，リアルタイム OS **用のタスクとして指定されました。**inline **指定は無視されます。**

W2691: Startup routine for RTOS task cannot be inlined, 'inline' is ignored.

リアルタイム OS **用のタスクとして指定された関数にインライン展開を指定することはできません。**inline **指定は無視されます。**

W2699: Function '*function*' is undefined, previously specified GP symbol for rtos_interrupt is ignored.

gp **シンボル指定のある割り込みハンドラとして指定された関数が，ファイル内で定義されていません。割り込みハンドラ指定は無視されます。**

W2700: cannot specify GP symbol, function '*function*' is already defined

定義済みの関数に対して gp **シンボルを指定することはできません。**gp **シンボル指定は無効となります。**

W2703: GP symbol is not specified for RTOS interrupt function '*function*'

リアルタイム OS **用割り込みハンドラとして指定された関数** *function* **に対して，**gp **シンボルが指定されていません。**

W2704: Function '*function*' is undefined, previously specified 'rtos_task' is ignored.

リアルタイム OS **用のタスクとして指定された関数** *function* **がファイル内で定義されていません。**rtos_task **指定は無視されます。**

W2710: Section '*section1*' is already specified for '*symbol*'. '*section2*' is ignored.

シンボル *symbol* **に対して，セクション・ファイル内で** *section1* **がすでに指定されています。あとから指定された** *section2* **は無視されます。**

W2711: Different section is specified for '*symbol*' in source file (*section1*) and section file (*section2*). Source file specification is ignored.

シンボル *symbol* に対して，ソース・ファイルにおけるセクション指定（*section1*）とセクション・ファイルにおけるセクション指定（*section2*）が異なっています。ソース・ファイルにおけるセクション指定（*section1*）は無視されます。

W2730: Block interrupt function cannot be installed, 'inline' is ignored.

割り込み禁止として宣言された関数には，“ inline **”を指定することはできません。**inline **指定は無視されます。**

W2731: Block interrupt attribute is specified for function '*function*', previously specified 'inline' is ignored.

“ inline **” 指定されている関数 *function* が割り込み禁止として指定されました。**inline **指定は無視されます。**

W2740: '#pragma text function-name' must be placed before the function's definition. '#pragma text *function*' is ignored.

text **セクション指定は関数定義の前で行わなければなりません。関数 *function* に対する** text **セクション指定は無視されます。**

W2741: Function specified as 'direct' can not be allocated in text. '#pragma text *function*' is ignored.

直接配置（direct**）で割り込みに指定されている関数に対し，**text **セクション指定することはできません。関数 *function* に対する** text **セクション指定は無視されます。**

W2742: Function allocated in text can not be specified as 'direct'. Previously specified '#pragma text *function*' is ignored.

text **セクション指定された関数を，直接配置（**direct**）で割り込みに指定することはできません。すでに指定されている関数 *function* に対する** text **セクション指定は無視されます。**

W2743: Function allocated in text can not be inlined. '#pragma inline *function*' is ignored.

text **セクション指定された関数には，“** inline **” を指定することはできません。関数 *function* への** inline **指定は無視されます。**

W2744: Function allocated in text can not be inlined. Previously specified '#pragma inline *function*' is ignored.

inline **指定されている関数に対し，**text **セクション指定されました。すでに指定されている関数 *function* への** inline **指定は無視されます。**

W2748: Section name is not specified.

#pragma section **のセクション名指定で，“ " ” の間にセクション名が指定されていません。セクション名指定がないものとみなし，指定した属性の予約セクションに割り当てます。**

W2761: unrecognized specfier '*specifier*', ignored

指定子 *specifier* は認識されません。この指定子は無視されます。

W2780: result of comparison is always false

比較式が常に偽になります。

W2782: statement with no effect

この構文は無効です。

W2784: Conversion may lose significant digit

データが失われている可能性があります。

W5009: sorry, not implemented option '*option*', ignored

オプション *option* は現在サポートされていません。無視されます。

W5501: The section of variable 'symbol' was changed from '*old*' to '*new*'.

変数 symbol **のセクションを** *old* **から** *new* **に変更しました。**

W5502: The size of variable '*symbol*' was changed from *old* to *new*.

変数 *symbol* **のサイズを** *old* **から** *new* **に変更しました。**

W5503: The alignment of variable '*symbol*' was changed from old to new.

変数 *symbol* **のアライメントを** *old* **から** *new* **に変更しました。**

W5504: The initial value of variable '*symbol*' was changed.

変数 *symbol* **の初期値を変更しました。**

W6101: immediate for shift operator is out of range

シフト命令に対し指定されたイミーディエトの値が指定可能な値の範囲を越えています。下位の有効な桁数分だけが指定されたものとみなし，処理を続行します。

W6102: first argument of _rcopy() is illegal

コピー・ルーチン _rcopy **の第一引数が不正です。**

W7101: sorry, not implemented option '*option*', ignored

オプション *option* は現在サポートされていません。無視されます。

W7102: redeclaration of '*symbol*'

symbol **が再宣言されています。**

W7103: Symbol '*symbol*' has different size (*num*1 and *num*2).

データ・シンボル *symbol* **に関して異なるサイズ（***num*1 **と** *num*2**）のマージが発生しました。**

W7104: Symbol '*symbol*' has different alignment size (*num*1 and *num*2). Changed to least common multiple value (*num*3).

データ・シンボル *symbol* **に関して異なるアライメント・サイズ（***num*1 **と** *num*2**）のマージが発生しました。最大公倍数** *num*3 **に変更します。**

W7105: cannot hide symbol '*symbol*'

シンボル *symbol* **をハイド化できません。**

W7106: Argument type mismatch is detected where '*caller*' calls '*callee*'.

インライン展開時に *caller* と *callee* の引数の型に違いがあります。変換できる場合は，定義での型に変換し，できない場合，インライン展開は無視されます。

W7107: Return value type mismatch is detected where '*caller*' calls '*callee*'.

インライン展開時に *caller* と *callee* の戻り値の型に違いがあります。変換できる場合は，呼び出し側の型に変換し，できない場合，インライン展開は無視されます。

B.1.3 アセンブラ

E3200: illegal alignment value

整列条件の指定に誤りがあります。

E3201: illegal character

扱うことのできない文字が現れました。

E3202: illegal expression

式の構成に誤りがあります。

E3203: illegal expression (*string*)

式の要素 *string* に誤りがあります。

E3204: illegal expression (-label)

（- ラベル）の形式の式が用いられています。

E3205: illegal expression (-label - label)

（- ラベル - ラベル）の形式の式が用いられています。

E3206: illegal expression (label + label)

（ラベル + ラベル）の形式の式が用いられています。

E3207: illegal expression (labels have different reference types)

異なる形式のラベル参照（#label，label，および $label）の間に演算が指定されています。

E3208: illegal expression (labels in different sections)

異なるセクションに属するラベル間に演算が指定されています。

E3209: illegal expression (labels must be defined)

ラベル同士の演算は同一ファイル内に定義してください。

E3210: illegal expression (not + nor -)

+，- 以外の演算が用いられています。

E3211: floating exception(*function*)

as850 が内部的に用いている浮動小数点演算ライブラリの関数 *function* において，浮動小数点演算に誤りがあります。

E3212: symbol already defined as label

指定されたシンボルはすでにラベルとして定義されています。

E3213: label *identifier* redefined

ラベル *identifier* が複数回定義されています。

E3214: *identifier* redefined

***identifier* が複数回定義されています。**

E3215: illegal operand (access width mismatch)

オペランドに異なるアクセス幅の内部周辺 I/O レジスタを指定しています。

E3216: illegal operand (cannot read I/O register which does not have read access)

オペランドに指定した内部周辺 I/O レジスタは読み出し禁止です。

E3217: illegal operand (cannot use bit I/O register)

オペランドに内部周辺 I/O レジスタのフラグのビットを指定することはできません。

E3218: illegal operand (cannot write I/O register which does not have write access)

オペランドに指定した内部周辺 I/O レジスタは書き込み禁止です。

E3219: illegal operand (inconsistent bit position)

ビット操作命令で指定したビット位置が矛盾しています。

E3220: illegal operand (*identifier* is reserved word)

名前に予約語 *identifier* が用いられています。

E3221: illegal operand (label - label)

分岐命令に対し，（ラベル - ラベル）の形式の式が指定されています。

E3222: illegal operand (label not allowed)

オペランドにラベルを指定することのできない命令に対し，ラベルが指定されています。

E3223: illegal operand (label not allowed for setf/shl...)

setf **命令，またはシフト命令に対し，ラベルが指定されています。**

E3224: illegal operand (label reference for jmp must be #label)

jmp **命令に対し，絶対アドレス参照（**#label**）以外のものが指定されています。**

E3225: illegal operand (must be evaluated positive or zero)

式の評価結果が負になりました。

E3226: illegal operand (must be even displacement)

奇数のディスプレースメントが指定されています。

E3227: illegal operand (must be immediate, label or symbol for hi/lo/hi1)

hi，lo，**および** hi1 **に対し，イミーディエト，ラベル，またはシンボル以外のものが指定されています。**

E3228: illegal operand (must be register)

レジスタ以外のものが指定されています。

E3229: illegal operand (needs base resister)

ベース・レジスタを指定する必要があります。

E3230: illegal operand (range error in displacement)

ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

E3231: illegal operand (range error in immediate)

イミーディエトとして指定された値が指定可能な値の範囲を越えています。

E3232: illegal operand (.local parameter)

.local **疑似命令に指定されたパラメータが不正です。**

E3233: illegal operand (local symbol parameter)

.local 疑似命令に指定されたパラメータがシンボルではありません。

E3234: illegal operand (macro parameter)

.macro 疑似命令に指定されたパラメータが不正です。

E3235: illegal operand (macro name)

.macro 疑似命令に定義されたマクロ名が不正です。

E3236: illegal operand (macro argument)

マクロ呼び出しに指定されたパラメータが不正です。

E3237: illegal operand (.irepeat argument)

.irepeat 疑似命令に指定された引数が不正です。

E3238: illegal operand (.irepeat parameter)

.irepeat 疑似命令に指定されたパラメータが不正です。

E3239: illegal operand (can not use r0 as source in V850E mode)

V850Ex コア指定時には，ソース・オペランドに r0 を指定することはできません。

E3240: illegal operand (can not use r0 as destination in V850E mode)

V850Ex コア指定時には，デスティネーション・オペランドに r0 を指定することはできません。

E3241: illegal operand (too many registers)

pushm ／ popm 命令に指定されたレジスタ数が多すぎます。

E3242: illegal operand (label is already defined on *section*)

.option sdata ／ .option data に指定したラベルは，すでに *section* セクションに定義されています。

E3244: illegal origin value(*value*)

.org 疑似命令において値（*value*）の指定に誤りがあります。

E3245: *identifier* is reserved word

予約語を用いることのできない場所において予約語 *identifier* が用いられています。

E3246: illegal section

セクション中に記述することのできない命令が記述されています。

E3247: illegal size value

サイズの指定に誤りがあります。

E3248: illegal symbol reference ($symbol)

シンボルに対し“$”，または“#”が指定されています。

E3249: illegal syntax

構成に誤りがあります。

E3250: illegal syntax *string*

string の構成に誤りがあります。

E3251: illegal id value

指定された ID **値が不正です。整数を指定してください。**

E3252: id already defined as symbol "*identifier*"

指定された ID **値は，シンボル名“** *identifier* **”としてすでに予約されています。**

E3253: symbol "*identifier*" already defined as another id

指定されたシンボル名“ *identifier* **”は，異なる** ID **値ですでに予約されています。**

E3254: can not reference .ext_func symbol "*identifier*"

.ext_func **疑似命令を用いて指定されたシンボルは，分岐命令以外で参照することはできません。**

E3255: cannot access for I/O register bit number "*I/O register*''

I/O **レジスタ名“** *I/O register* **”に指定したビット番号が間違っています。**

E3258: cannot access I/O register(''*I/O register*'')

オペランドに指定した内部周辺 I/O **レジスタ“** *I/O register* **”はアクセス禁止です。**

E3259: can not use r1 as destination in mul/mulu.

mul ／ mulu **命令のデスティネーション・レジスタにはアセンブラ予約レジスタ（**r1**）を指定できません。**

E3260: token too long

トークンの長さが限界を越えています。

限界値は 1037 **です。**

E3261: illegal condition code.

指定された条件コードが不正です。

adf,sbf **命令【**V850E2**】の条件コードに** 0xd **は指定できません。**

F3500: too many files

複数のファイルを指定することはできません。

F3501: illegal bit width

.byte，.hword，**または** .word **疑似命令においてビット幅の指定に誤りがあります。**

F3502: illegal file name (must be .s file)

入力ファイルの拡張子が不正です。拡張子は .s **としてください。**

F3503: can not open file *file*

ファイル *file* **をオープンできません。**

F3504: illegal section kind

.section **疑似命令においてセクションの種類の指定に誤りがあります。**

F3505: memory allocation fault

メモリが足りません。

F3506: memory allocation fault (*string*)

内部データ領域（*string***）の確保に失敗しました。**

F3507: overflow error (*string*)

式の処理において作業領域が足りなくなりました。単純な式に変更してください。

F3508: *identifier* undefined

定義されていない識別子 *identifier* が参照されています。

F3509: illegal pseudo(*string*) found

予期しない疑似命令 *string* が見つかりました。

F3510: *string* unexpected

***string* 疑似命令に対応する疑似命令が存在しません。**

F3511: *string* unmatched

条件アセンブル疑似命令において対応する疑似命令 *string* が存在しません。

F3512: .if, .ifn, etc. too deeply nested

条件アセンブル疑似命令が 17 回以上ネストして用いられています。

F3513: unexpected EOF in *string*

***string* 疑似命令に対応する** .endm **疑似命令が存在しません。**

F3514: paramater table overflow

実パラメータが 33 個以上用いられています。

F3515: string not in .repeat/.irepeat

***string* 疑似命令が繰り返しアセンブル疑似命令に囲まれていません。**

F3516: local symbol value overflow

.local **疑似命令により自動生成されたシンボルが限界数（65536）を越えました。**

F3517: *string* nest over

***string* が 9 回以上ネストして用いられています。**

F3518: unreasonable macro_call nesting

マクロ本体内で現在定義中のマクロの呼び出しが行われました。

F3519: argument mismatch

マクロ呼び出しの引数指定が不正です。

F3520: $ must be followed by defined symbol

“ $ ” のあとにシンボル以外の識別名，または未定義シンボル名が指定されています。

F3521: too many errors

致命的なエラーの数が 30 に達しました。アセンブルを中止します。

F3522: unknown cpu type

ターゲット・デバイス指定がないため，アセンブルできません。

F3523: duplicated cpu type

オプション，または疑似命令でターゲット・デバイス指定が重複しています。

F3524: can not find devicefile

指定されたターゲット・デバイスに相当するデバイス・ファイルがないか，デバイス指定が間違っている，またはデバイス指定がありません。

F3525: illegal cpu family

指定されたデバイス・ファイルが V850 **マイクロコントローラ用ではありません。**

F3526: devicefile version mismatch, cannot use version *version*

指定されたデバイス・ファイルのバージョンが不正です。バージョン *version* **は指定できません。**

F3527: nested command file *file*

コマンド・ファイル *file* **がネストしています。ネストはできません。**

F3528: .tidata.byte/.tibss.byte size overflow(size > 128).

.tidata.byte **セクション，**.tibss.byte **セクションのサイズの合計が** 128 **バイトを越えています。**

F3529: .tidata.word/.tibss.word size overflow(size > 256).

.tidata.word **セクション，**.tidata.word **セクションのサイズの合計が** 256 **バイトを越えています。**

F3530: .tidata/.tibss size overflow(size > 256).

.tidata.byte **セクション，**.tibss.byte **セクション，**.tidata.word **セクション，**.tibss.word **セクション，**.tidata **セクション，**.tibss **セクションのサイズの合計が** 256 **バイトを越えています。**

F3531: too many symbols

1 **ファイルに記述できるシンボル数を越えました。記述できるシンボル数の限界は，アセンブラが内部で登録するものを含め，**16,777,215 **です。**

F3532: illegal object file (*string*)

リンク可能なオブジェクト・ファイルを生成する段階で，ファイル・システムに依存するエラーが発生しました。

W3000: .option az_info_*kind* unmatched, ignored.

.option az_info_*kind* **が不正な位置で指定されています。無視されます。**

W3001: too many actual parameter.

マクロ呼び出し時に指定された実パラメータが多すぎます。

W3002: can not use *option1* with *option2*, *option2* ignored.

option1 **オプションと** *option2* **オプションを同時に指定することはできません。***option2* **オプションは無視されます。**

W3003: "*option*'' option needs argument, ignored.

option **オプションには引数の指定が必要です。オプション指定は無視されます。**

W3004: illegal "*option*'' option's value, ignored.

option **オプションに指定された値が不正です。オプション指定は無視されます。**

W3005: illegal "*option*'' option's symbol "*symbol*'', ignored.

option **オプションに指定されたシンボル** *symbol* **が不正です。オプション指定は無視されます。**

W3006: illegal "*option*" option's argument, ignored.

option オプションに指定された引数が不正です。オプション指定は無視されます。

W3007: *option*, -cpu mismatch. ignore -cpu. output *core* common object.

core コア共通のオブジェクト生成を指定する *option* オプションと，-cpu オプションで指定したデバイス・ファイルに不整合があります。-cpu オプション指定を無視し，*core* コア共通オブジェクトを生成します。

W3008: *option* option is not supported for *core* core.

option オプションは *core* コアではサポートしていません。オプション指定は無視されます。

W3009: can not find programmable peripheral I/O registers, ignored -bpc option.

プログラマブル周辺 I/O レジスタの情報が存在しません。-bpc オプションは無視されます。

W3010: illegal displacement in *inst* instruction.

ディスプレースメントの値が指定可能な値の範囲を越えています。下位の有効な桁数分だけが指定されたものとみなし，アセンブルを続行します。

W3011: illegal operand (range error in immediate).

イミーディエトの値が指定可能な値の範囲を越えています。下位の有効な桁数分だけが指定されたものとみなし，アセンブルを続行します。

W3012: hword overflow.

.hword ／ .shword に指定した値が指定可能な値の範囲を越えています。下位の有効な桁数分だけが指定されたものとみなし，アセンブルを続行します。

W3013: *register* used as *kind* register.

レジスタ (r0)，アセンブラ予約レジスタ (r1)，またはマスク・レジスタ機能使用時に，マスク・レジスタ（r20，または r21）が，*kind* レジスタとしてオペランドに指定されています。

W3014: illegal list value, ignored.

prepare ／ dispose 命令のレジスタ・リストに指定した値が不正です。下位の有効な桁数分だけが指定されたものとみなし，アセンブルを続行します。

W3015: illegal register number, ignored.

prepare ／ dispose 命令のレジスタ・リストに指定したレジスタが不正です。不正なレジスタを無視して，アセンブルを続行します。

W3016: illegal operand (access width mismatch).

オペランドに，異なるアクセス幅の内部周辺 I/O レジスタを指定しています。

W3017: base register is ep(r30) only.

sld ／ sst 命令のベース・レジスタに ep 以外を指定しています。

W3018: illegal regID for *inst*.

inst 命令に指定した番号のシステム・レジスタはアクセス禁止です。

W3019: illegal operand (immediate must be multiple of 4).

オペランドに指定した値は 4 の倍数である必要があります。端数を切り捨てて，アセンブルを続行します。

W3020: duplicated cpu type, ignored .option cpu.

-cpu オプションによる指定と .option 疑似命令による指定が異なっています。-cpu オプションを優先し，.option 疑似命令によるターゲット・デバイス指定は無視されます。

W3021: *string* already specified, ignored.

string が，以前に指定したレジスタ数と異なる数で指定されています。すでに指定されている数を使用します。この指定は無視されます。

W3022: duplicated *option*, ignored.

option が，複数回指定されています。すでに指定されているオプションを使用します。この指定は無視されます。

W3023: BPC value is out of range(0x0-*value*), ignored -bpc option.

-bpc に指定された値が，デバイスの許容する範囲 (0x0-*value*) 外です。指定された値は無視して，デバイスの初期値を使用します。

W3024: sorry, *option* option not implemented, ignored.

option オプションはインプリメントされていません。無視されます。

W3025: sorry, *option* option not implemented, ignored.

option オプションはインプリメントされていません。無視されます。

W3026: illegal register number, aligned odd register(rXX) to be even register(rYY).

奇数番号の付いたレジスタ（r1，r3，…，r31）を指定しました。

指定できる汎用レジスタは，偶数番号の付いたレジスタ（r0，r2，r4，…，r30）だけです。

偶数番号の付いたレジスタ（r0，r2，r4，…，r30）を指定したとして，アセンブルを続行します。

B.1.4 リンカ

E4231: ';' is expected at the end of directive.

ディレクティブの終わりには“;（セミコロン）”が必要です。

E4232: '}' is expected.

“}”が必要です。

E4233: name is expected at the beginning of directive.

ディレクティブは名前（セグメント名／セクション名／シンボル名）で始めてください。

E4234: section name is expected at the beginning of section directive.

セクション・ディレクティブはセクション名で始めてください。

E4235: ':', '=' or '@' is expected to follow name.

ディレクティブの始まりの名前の後ろには“:”,“=”,“@”のいずれかが必要です。

E4236: '=' is expected to follow section name.

出力セクション名の後ろには“=”が必要です。

E4237: too many '}'.

“{”に対応する“}”が多すぎます。

E4238: illegal character (*number*).

リンク・ディレクティブに不正な文字（*number*）が存在しています。

E4239: *string* needs effective parameter.

***string* には有効なパラメータが必要です。**

E4240: *string* is illegal in segment directive.

セグメント・ディレクティブにおいて *string* を指定することはきません。

E4241: *string* is illegal in section directive.

セクション・ディレクティブにおいて *string* を指定することはできません。

E4242: *string* is illegal in symbol directive.

シンボル・ディレクティブにおいて *string* を指定することはできません。

E4243: *string* is illegal in file specification field.

ファイル名を指定する部分に *string* を指定することはできません。

E4244: *string* illegal in segment name field.

セグメント名を指定する部分に *string* を指定することはできません。

E4245: *string* specified to segment "*segment*" more than once in same or other directive.

***string* がセグメント *segment* に対し，同じセグメント・ディレクティブ，または別のセグメント・ディレクティブにおいて複数回指定されています。**

E4246: *string* specified to section "*section*" more than once in same or other directive.

string **がセクション** seciton **に対し，同じセクション・ディレクティブ，または別のセクション・ディレクティブにおいて複数回指定されています。**

E4247: *string* specified to symbol "*symbol*" more than once in same or other directive.

string **がシンボル** *symbol* **に対し，同じシンボル・ディレクティブ，または別のシンボル・ディレクティブにおいて複数回指定されています。**

E4248: segment"*segment*" already defined.

セグメント *segment* **はすでに定義されています。**

E4249: section "*section*" already defined at line(*number*).

セクション *section* **は** *number* **行目においてすでに定義されています。**

E4250: symbol "*symbol*" already defined at line(*number*).

シンボル *symbol* **は** *number* **行目においてすでに定義されています。**

E4251: illegal segment type "*string*".

セグメント・タイプとして指定することのできない *string* **が指定されています。**

E4252: illegal section type "*string*".

セクション・タイプとして指定することのできない *string* **が指定されています。**

E4253: illegal section attribute 'character'

セクション属性として指定することのできない character **が指定されています。**

E4254: *string* in segment directive of non LOAD segment is illeagal.

セグメント・タイプに LOAD **を指定してないセグメント・ディレクティブにおいて** *string* **を指定することはできません。**

E4267: unknown symbol kind "*string*".

シンボル種別として指定することのできない *string* **が指定されています。**

E4268: symbol kind "*string*" specified more than once in same or other directive.

シンボル種別 *string* **が同じディレクティブ，または別のディレクティブにおいて複数回指定されています。**

E4271: section attribute 'attribute' of section "*section*" and segment attribute 'attribute' of segment "*segment*" do not match.

セクション *section* **のセクション属性** attribute **とこのセクションの割り付けが指示されているセグメント** *segment* **のセグメント属性が一致しません。**

[対処方法]

セクション属性 G **は無視し，セクション属性** A，W，X **がそれぞれセグメント属性** R，W，X **に相当するものとして一致するようにしてください。**

F4001: can not open command file "*file*".

コマンド・ファイル *file* をオープンできません。

F4002: can not open input file "*file*".

入力ファイル *file* を開けません。

F4003: can not open output file "*file*".

出力ファイル *file* を開けません。

F4004: can not create output file "*file*".

出力ファイル *file* を作成できません。

F4005: can not open directive file "*file*"

ディレクティブ・ファイル *file* を開けません。

F4006: can not get size of directive file "*file*".

ディレクティブ・ファイル *file* のサイズの取得に失敗しました。

F4007: can not read directive file "*file*"

ディレクティブ・ファイル *file* が読み込めません。

F4008: can not truncate output file "*file*" to have size(*number*).

出力ファイル *file* のサイズを *number* バイトに変更できません。

F4009: can not seek output file "*file*".

出力ファイル *file* をシークできません。

F4010: can not write output file "*file*".

出力ファイル *file* の書き込みができません。

F4011: can not find divicefile "*string*".

デバイス・ファイル *string* が見つかりません。

F4012: illegal device file "*string*".

デバイス・ファイル *string* が不正です。

F4013: can not open device file "*string*",

デバイス・ファイル *string* をオープンできません。

F4014: can not read device file "*string*".

デバイス・ファイル *string* の読み込みができません。

F4015: illegal object file (Error Number:*number*).

オブジェクト・ファイルが不正です。

F4031: illegal ELF version.

指定されたオブジェクト・ファイルの ELF **形式の版が** ld850 **の扱うことのできる版ではありません。**

F4032: illegal target machine type.

入力ファイルのタイプが ld850 **の扱うことのできるタイプではありません。**

F4033: illegal target machine class.

入力ファイルのクラスが ld850 **の扱うことのできるクラスではありません。**

F4034: illegal target machine byte order.

入力ファイルのバイト・オーダが ld850 **の扱うことのできるバイト・オーダではありません。**

F4035: illegal ELF file type, must be relocatable or shared library file.

入力ファイルとして扱うことのできるオブジェクト・ファイルのファイル・タイプは，リロケータブル・ファイル，または共有ライブラリ・ファイルのみです。

F4036: unknown format type file "*file*".

指定されたファイル *file* **は不正なファイル形式を持っています。**

F4037: illegal devicefile. different family "*number*".

デバイス・ファイルが違います。ファミリ *number* **が異なっています。**

F4038: "*file*" is not executable file.

-zf **オプションで指定したブート側ファイル** *file* **は，実行可能形式ではありません。**

ld850 **の出力した実行可能形式ファイルを指定してください。**

F4039: "*file*" is rom packed file.

-zf **オプションで指定したブート側ファイル** *file* **は，**ROM **化されています。**

ROM **化を行う以前の，**ld850 **の出力した実行可能形式ファイルを指定してください。**

F4051: fail to get symbol name *string*.

シンボル名文字列の取得に失敗しました。

F4052: failed to get *number* th symbol name *string*.

number **番目のシンボル名文字列の取得に失敗しました。**

F4053: symbol "*symbol*" has unknown binding class(*number*).

シンボル *symbol* **は，不正なバインディング・クラス** *number* **を持っています。**

F4054: weak symbol "*symbol*" not supported.

WEAK **のバインディング・タイプを持つシンボル** *symbol* **はサポートしていません。**

F4058: symbol "*symbol*" multiply defined.

シンボル *symbol* は，多重に定義されています。

[注意事項]

PM+ を使用している場合，およびスタート・アップ・ファイルをプロジェクトのソース・ファイルに登録しているような場合は次のようになります。

```
symbol "__start" multiply defined.
```

PM+ を使用した場合は，[コンパイラ共通オプションの設定] ダイアログの “ スタートアップファイル ” に何も指定しないと，デフォルトのスタート・アップ・モジュールが指定されたと見なされます。

次のいずれかの対処を行ってください。

ソース・ファイルの登録から，スタート・アップ・ファイルを外し，[コンパイラ共通オプションの設定] ダイアログの “ スタートアップファイル ” にスタート・アップ・ファイルを指定する。

ダミーのアセンブリ言語ファイルを作成し，アセンブルして *.o ファイルを作成して，[コンパイラ共通オプションの設定] ダイアログの “ スタートアップファイル ” に指定する。

F4059: linking of symbol "*symbol*" in sdata of sbss attribute section in "file1" and in other attribute section in "file2" is attemped.

file1 内で定義されたシンボル *symbol* と file2 内で定義された同じ名前のシンボルのセクション配置に矛盾が生じました。

F4060: can not find entry point symbol "*symbol*" specified with "-e" option.

-e オプションで指定されたシンボル *symbol* が見つかりません。

F4063: ".ext_func ID, symbol" is already defined as ".ext_func ID, symbol" in other file.

.ext_func 疑似命令で指定されたシンボル名と ID 値に矛盾があります。

F4065: too many symbols.

シンボル数が限界値を越えました。-r（-ro）オプションで作成するリロケータブル・オブジェクト・ファイルのシンボル限界値は 16777125 です。

F4101: failed to get section name *string* table section.

セクション名のストリング・テーブル・セクションの取得に失敗しました。

F4102: fail to get symbol name *string* table section.

ストリング・テーブル・セクションの取得に失敗しました。

F4103: failed to get section header.

セクション・ヘッダ取得に失敗しました。

F4104: failed to get section name *string*.

セクション名の取得に失敗しました。

F4106: section "*section*" has unknown section type(*number*).

セクション *section* は，不正なセクション・タイプ *number* を持っています。

F4107: can not get raw data of section "*section*".

セクション *section* のロウ・データの取得に失敗しました。

F4109: interrupt function section "*section*" is already defined.

割り込みハンドラ・セクション *section* は，すでに定義されています。ld850 **では，指定されたデバイス・ファイルから割り込みハンドラ・セクションを自動生成するため，リンク・ディレクティブへの記述は不要です。**

F4110: special section "*section*" is already defined.

特殊セクション *section* は，すでに定義されています。

F4112: illegal "*section*" section size.

セクション *section* のサイズが不正です。

F4155: can not find GP-symbol in segment "*segment*" of illegal label reference for local symbol in file "*file2*" (section:*section2*, offset:*offset*, type:*relocation type*). local symbol is allocated in section "*section1*"(file:*file1*).

セグメント *segment* に GP **シンボルが指定されていません，またはローカル・シンボルの配置 / 参照方法に矛盾があります。**

[補足]

変数定義に対して #pragma section **指定で配置するセクションを変更した場合には,他のファイルにあるその変数に対する** extern **宣言に対しても，同じ方法でセクションを変更してください。ローカル・シンボルは，ファイル *file1* の *section1* に配置されています。ローカル・シンボルに対する参照は，ファイル *file2* のセクション *section2* のオフセット *offset* に存在します。**

F4156: can not find GP-symbol in segment "*segment*" of illegal label reference for "*symbol*" in file "file2" (section:*section2*, offset:*offset*, type:*relocation type*). "*symbol*" is allocated in section "*section1*"(file:file1).

セグメント *segment* に GP **シンボルが指定されていません，またはシンボル *symbol* の配置／参照方法に矛盾があります。**

[補足]

変数定義に対して #pragma section **指定で配置するセクションを変更した場合には,他のファイルにあるその変数に対する** extern **宣言に対しても，同じ方法でセクションを変更してください。シンボル *symbol* は，ファイル *file1* の *section1* に配置されています。シンボル *symbol* に対する参照は，ファイル *file2* のセクション *section2* のオフセット *offset* に存在します。**

F4157: can not find GP-symbol in section "*section*" of file "file1"or illegal label reference for symbol "*symbol*" in file "*file2*" (section:*section2*, offset:*offset*, type:*relocation type*). "*symbol*" is allocated in section "*section1*"(file:*file1*).

ブート側実行形式ファイル *file1* のセクション *section* に GP **シンボルが指定されていません，またはシンボル *symbol* の配置／参照方法に矛盾があります。**

[補足]

変数定義に対して #pragma section **指定で配置するセクションを変更した場合には,他のファイルにあるその変数に対する** extern **宣言に対しても，同じ方法でセクションを変更してください。シンボル *symbol* は，ファイル *file1* の *section1* に配置されています。シンボル *symbol* に対する参照は，ファイル *file2* のセクション *section2* のオフセット *offset* に存在します。**

F4158: relocated value(*value*) of relocation entry(symbol: symbol, file: *file*, section: section, offset: *offset*, type: *relocation type*) for branch command become odd value.

分岐系のリロケーション・エントリ（シンボル *symbol*，ファイル *file*，セクション *section*，オフセット *offset*，リロケーション・タイプ *relocation type*）によってリロケートされた値 *value* が奇数になっています。

F4160: EP symbol is needed for using SIDATA/SEDATA segment.

ep **シンボルが生成されていません。**SIDATA ／ SEDATA **セグメントを使用する際には** EP **シンボルを生成してください。**

F4161: symbol "*symbol*" (output section *section1*) is too far from output section "*section2*".(value: *value*,file: *file*, input section: *section3*, offset: *offset*, type: *relocation type*).

出力セクション *section2* から出力セクション *section1* に配置されたシンボル *symbol* への分岐命令が，分岐範囲を越えています。分岐命令は，ファイル *file* 中のセクション *section3* のオフセット *offset* の位置に存在します。

F4162: output section "*section1*" is too far from ouptput section "*section2*". (value: *value*, file: *file*, input section: *section3*, offset: *offset*, type: *relocation type*).

出力セクション *section2* から出力セクション *section1* に配置されたローカル・シンボルへの分岐命令が，分岐範囲を越えています。分岐命令は，ファイル *file* 中のセクション *section3* のオフセット *offset* の位置に存在します。

F4163: output section "*section1*" overflowed or illeagal label reference for symbol "*symbol*" in file "*file2*" (value: *value*, file: *file*, input section: *section2*, offset: *offset*, type: *relocation type*). "*symbol*" is allocated in section "*section1*"(file: file1).

出力セクション *section1* がオーバーフローしました,またはシンボル *symbol* の配置されたセクションとラベル参照方法が不正です。*symbol* は，ファイル *file1* で *section1* に配置されています。不正な参照はファイル *file2* のセクション *section2* のオフセット *offset* に存在します。

F4164: output section "*section1*" overflowed or illeagal label reference for local symbol in file "*file2*" (value: *value*, file: file, input section: *section2*, offset: *offset*, type: *relocation type*). local symbol is allocated in section "*section1*"(file: *file1*).

出力セクション *section1* がオーバーフローしました，またはローカル・シンボルの配置されたセクションとラベル参照方法が不正です。ローカル・シンボルは，ファイル *file1* で *section1* に配置されています。不正な参照はファイル *file2* のセクション *section2* のオフセット *offset* に存在します。

F4165: can not reference extern symbol "*symbol*" by *string*.

フラッシュ／外 ROM **再リンク機能において，ブート領域作成時にフラッシュ領域側のシンボル *symbol* を *string* による参照ができません。**

F4166: no symbol information in boot file "*file*".

-zf **オプションで指定したブート側ファイル *file* にベースシンボル情報が含まれていません。ブート側ファイルのリンク時に** -ext_table **が指定されているかを確認してください。**

F4203: can not find archive member at offset(*offset*) specified in archive symbol table entry.

アーカイブ・シンボル・テーブル・エントリで指定されたオフセット *offset* の位置にアーカイブ・メンバが見つかりません。アーカイブ・シンボル・テーブルの内容が破壊されているおそれがあります。

F4204: library path length is too long. path maximum size is 576.

ライブラリ・パスが長すぎます。576 **文字以下にしてください。**

F4257: segment "*segment*" overflowed highest address of target machine.

セグメント *segment* の割り付けが指示された領域がターゲット・マシンにおける可能なメモリ空間の範囲を越えています。

[補足]

-Ximem_overflow=warning **オプションを指定することにより，このメッセージを警告にすることができます。**-rom_less **オプションを指定することにより，エラー発生原因が内蔵** ROM **に関連する場合のみ，このメッセージを消去できます。**

F4258: segment directive of segment "*segment*" needs *string*.

セグメント *segment* のセグメント・ディレクティブには *string* が必要です。

F4259: section directive of section "*section*" needs *string*.

セクション・ディレクティブには *string* が必要です。

F4260: symbol directive of symbol "*symbol*" needs *string*.

シンボル *symbol* のシンボル・ディレクティブには *string* が必要です。

F4263: start address(*number1*) of segment "*segment1*" overlaps previous segment "*segment2*" ended before address(*number2*).

セグメント *segment1* の先頭アドレス *number1* がアドレス *number2* の前までに割り付けられているセグメント *segment2* の領域にオーバラップしています。

[注意事項]

オーバラップするはずのないセグメントが，リンク時にオーバラップによるエラーが発生してリンクできない場合，リンク・ディレクティブの指定で，上位アドレスのセグメントが先に記述されていないかどうか確認してください。

次にエラーとなる例を示します。

例

```
DATA1: !LOAD ?RW V0x300000{
       .data  = $PROGBITS  ?AW;
       .sdata = $PROGBITS  ?AWG;
       .sbss  = $NOBITS    ?AWG;
       .bss   = $NOBITS    ?AW;
};

TEXT : !LOAD ?RX V0x100000 L0x100000{
       .text  = $PROGBITS ?AX .text;
};
```

セグメントのアドレスは，数字の小さい順に指定しなければなりません。したがって，セグメントは，下位のアドレスに割り当てるものから先に記述する必要があります。

F4264: start address(*number1*) of section "*section1*" overlaps previous section "*section2*" ended before address(*number2*).

セクション *section1* の先頭アドレス *number1* がアドレス *number2* の前までに割り付けられているセクション *section2* の領域にオーバラップしています。

F4265: start address(*number1*) of section "*section1*" oveflowed start address(*number2*) of segment "*segment*".

セクション *section* の先頭アドレス *number1* が属するセグメント *segment* の先頭アドレス *number2* よりも手前に割り付けられています。

F4266: memory size(*number1*) of segment "*segment*" overflowed specified or default maximum memory size(*number2*).

セグメント *segment* のメモリ・サイズ *number1* が，明示的に指定された最大メモリ・サイズ，またはデフォルトの最大メモリ・サイズを越えています。

F4276: TP symbol "*symbol1*" specified as GP symbol "*symbol2*"'s base symbol is not found.

gp **シンボル *symbol2* のベース・シンボルとして指定された** tp **シンボル *symbol1* が見つかりません。**

F4279: end address of section "*section*" overflowed maximum memory address(*number*).

セクション *section* の終端アドレスが最大メモリ・サイズ *number* をオーバーフローしました。

F4280: end address of segment "*section*" overflowed maximum memory address(*number*).

セグメント *segment* の終端アドレスが最大メモリ・サイズ *number* をオーバーフローしました。

F4282: segment "*segment*" (*number1-number2*) overflowed highest or lowest address of internal memory (*number3-number4*).

セグメント *segment* の割り付けが指示された領域（*number1-number2*）がターゲット・マシンにおける可能なメモリ空間の範囲（*number3-number4*）を越えています。

[補足]

-Ximem_overflow=warning **オプションを指定することにより，このメッセージを警告にすることができます。**-rom_less **オプションを指定することにより，エラー発生原因が内蔵** ROM **に関連する場合のみ，このメッセージを消去できます。**

F4286: section "*section*" must be specified in link directive.

セクション *section* は，ディレクティブ・ファイルで指定されている必要があります。

F4287: description of section "*section*" in mapping directive is illegal.

ディレクティブ・ファイル中のセクション *section* の記述が不正です。

F4333: can not allocate memory (builtin new error).

メモリ領域の確保に失敗しました。

F4351: unknown option "*string*".

不正なオプション *string* が指定されています。

F4353: '-' is illeagal.

“ - ” のみを指定することはできません。

F4355: nesting of command file "*file*" in command file is not supported.

コマンド・ファイルにおいてコマンド・ファイル *file* が指定されています。コマンド・ファイルのネスティングはサポートしていません。

F4356: "*string1*" option is illegal when "*string2*" option is specified.

***string2* オプションンが指定された場合，*string1* オプションを指定することはできません。**

F4359: "*string*" option needs hexadecimal argument.

***string* オプションは，**16 **進数の引数を必要とします。**

F4361: illegal character (*number*) in "*string*" field.

オプション *string* に指定に不正な文字 *number*(ASCII **コード**) **が用いられています。**

F4363: unknown cpu type.

ターゲット・デバイスを指定してください。

[補足]

このメッセージは,リンク可能なオブジェクト･ファイル生成中に as850 **の** -cn ／ -cnv850e ／ -cnv850e2 **オプションを指定されたファイルのみをリンクして，実行可能なオブジェクト・ファイルを生成しようとした場合に出力されます。**

F4364: duplicated cpu type.

ターゲット・デバイスが重複しています。リンクするオブジェクト・ファイルで，異なるターゲット・デバイスが指定されています。

F4369: "*string1*" is illegal when "*string2*" option is specified.

オプション *string2* の指定時には "*string1*" は指定できません。

F4370: "*string1*"option needs "*string2*" option.

***string1* オプションは *string2* オプションを必要とします。**

F4374: "*string*" option's value overflowed.

***string* オプションに指定した値がオーバーフローしました。**

F4404: symbol table overflow.

シンボル・テーブルの領域が足りなくなりました。

F4409: sorry, shared library not supported.

共有ライブラリはサポートしていません。

F4411: multiple inclusion of same file attempted, ignored.

同じファイルが入力ファイルとして複数回指定されています。

[注意事項]

リンカのフラッシュ対応オブジェクトの生成で，-zf で指定したブート・オブジェクト・ファイル名がリンカの入力ファイル名と同じである場合，このエラーとなります。この場合は，どちらかのファイル名を変更してください。

F4412: command line length is too long. path maximum size is 512.

コマンド・ラインに指定した文字が多すぎます。指定可能最大文字数は 512 バイトです。

F4413: *file* has different .ext_ent_size.

入力ファイル *file* は，他と異なる .ext_ent_size **を指定されています。入力ファイル中の** .ext_ent_size **指定を統一してください。**

F4414: CallTBasePointer(CTBP) is not set. CTBP must be set when compiler option" -Ot" (or" -Xpro_epi_runtime=off") is not specified.

CALLT **ベース・ポインタ（**CTBP**）が設定されていません。プロローグ／エピローグ・ランタイムのオプション設定が未使用（**-Xpro_epi_runtime=off**），またはより高度な最適化（実行速度優先）**"-Ot" **指定時以外の場合には，**CTBP **を設定してください。**

[プロローグ／エピローグ・ランタイム呼び出しの注意]

プロローグ／エピローグ・ランタイムは，標準ライブラリに含まれています。標準ライブラリのリンク指定（-lc**）がない場合にも，このエラーとなりますので，ライブラリのリンク指定を確認してください。**

F4415: S-JIS code (*number1*, *number2*) is broken in *string*.

***string* に指定した** S-JIS **コードが不正です。**

F4451: multiple defined symbol.

```
symbol        defined file  previous defiend file
symbol        file1         file2
```

***file1* で指定されているシンボル *symbol* は *file2* ですでに定義されています。**

F4452: undefined symbol.

symbol referenced in "*file*"

ファイル *file* 内で参照されているシンボル *symbol* が定義されていません。

[注意事項]

(1) **ライブラリのリンク指定を行っているのに，ランタイム・ライブラリなどでこのエラーとなる場合は，ライブラリのリンク指定の順番を確認してください。**-l **オプションは，指定された時点で未解決な外部参照についてのみ，シンボル解決を行います。このオプションは，一般的に指定するアーカイバ・ファイルより，後ろに指定する必要があります。**

なお，-rescan **オプションを指定することにより，**-l **オプションで指定されたライブラリのリンク順によるシンボル未解決を防ぐことができます。**

次に変更例を示します。

例

```
-lm -lc a.o b.o c.o
↓
a.o b.o c.o -lm -lc
```

(2) **記憶域管理用ライブラリ（**calloc,malloc,free,realloc**）を使用しているときに次のエラー・メッセージが出力される場合は，ヒープ・メモリの確保を行ってください。**

F4452: undfined symbol.

__sysheap (refrenced in "heapcom.o(*Install Folder*\lib850\r22\libc.a)")

__sizeof__sysheap (refrenced in "heapcom.o(*Install Folder*\lib850\r22\libc.a)")

(3) **周辺機能レジスタ名が** undefined symbol **となる場合には，周辺機能レジスタ名を** extern **宣言して使用している場合が考えられます。周辺機能レジスタを使用する場合は，**extern **宣言部を削除し** #pragma ioreg **での指定をしてください。**

F4453: device file version mismatch, cannot use version *string*.

指定されたデバイス・ファイルのバージョンが不正です。バージョン *string* は指定できません。

F4454: cannot link V850E(2) common objects with V850(E) objects. "file" is V850E(2) common

次の組み合わせでは，リンクすることはできません。

ファイル *file* は V850Ex **コアの共通オブジェクト・ファイルです。**

- V850E1/V850ES **コアの共通オブジェクト・ファイルと** V850 **コアのオブジェクト・ファイル**
- V850E2 **コアの共通オブジェクト・ファイルと** V850 **コアのオブジェクト・ファイル**
- V850E2 **コアの共通オブジェクト・ファイルと** V850E1/V850ES **コアのオブジェクト・ファイル**

F4455: cannot link old_fcall objects with new_fcall object. "*file*" is old_fcall object

新関数呼び出し仕様のオブジェクト・ファイルと旧関数呼び出し仕様のオブジェクト・ファイルをリンクすることはできません。ファイル *file* は旧関数呼び出しのオブジェクト・ファイルです。

F4456: cannot link mask reg using objects with mask reg not using objects. "*file*" is mask reg using object.

マスク・レジスタを使用したオブジェクト・ファイルと使用していないオブジェクト・ファイルをリンク

することはできません。ファイル *file* はマスク・レジスタを使用したのオブジェクト・ファイルです。

F4457: input files have different BPC value.

BPC 値の異なるファイルが入力されています。

W4504: can not create output file "*file*".

出力ファイル *file* を作成できません。

W4555: symbol "*symbol*" has incompatible type in file "*file*".

シンボル *symbol* をリンクしようとしましたが,*file* において定義されている同名のシンボルとは型が異なっています。

W4556: symbol "*symbol*" has different size in file "*file*".

シンボル *symbol* をリンクしようとしましたが，*file* において定義されている同名のシンボルとはサイズが異なっています。

W4557: symbol "*symbol*" has different align-size in file "*file*".

シンボル *symbol* においてリンクが生じましたが，*file* において定義されている同名のシンボルとは整列条件が異なっています。

W4562: size zero sbss or bss attribute symbol "*symbol*".

sbss，または bss 属性セクションに割り付けられる *symbol* のサイズが 0 でした。

W4564: undefined global symbol *symbol* specified with "ext_func".

.ext_func 疑似命令で指定されたグローバル・シンボル *symbol* が定義されていません。

[補足]

次のいずれかの原因により出力されます。

(1) 関数 *symbol* が定義されていない場合

(2) 関数 *symbol* が static 指定されている場合

(3) *symbol* を 指定した .ext_func 疑似命令が関数定義したソースに記述されていない場合

なお，分岐テーブル中の該当する関数への分岐命令は未解決となります。

W4605: section "*section*" with section type (*section type*) not supported, ignored.

セクション・タイプ *section type* を持つセクション名 *section* のセクションは，ld850 においてサポートしていません。無視されます。

W4608: input files have different register modes. use -rc option for more information.

レジスタ・モードの異なるファイルが入力されています。-rc オプションを指定すると，より詳細な情報を出力します。

[注意事項]

32 レジスタ・モード以外のレジスタ・モードを指定していて，ソース・ファイルにアセンブリ言語ファイルが含まれるような場合，この警告メッセージが出力されることがあります。アセンブリ言語ファイルのレジスタ・モードの指定は，アセンブル・ファイル中に .option 疑似命令を使用して，レジスタ・モードを指定する必要があります。次の記述をアセンブリ言語ファイル中に記述してください。

22 レジスタ・モードを用いるとき：

```
.option reg_mode 5 5
```

26 レジスタ・モードを用いるとき：

```
.option reg_mode 7 7
```

W4611: "*string*" option overrides "*section*" section.

string オプションにより，*section* セクションを上書きします。

W4613: illegal flash mask option access (file:"*file*" address:num1 bit:num2).

オプション・バイトの設定不可能機能を設定しました。

W4614: section "*section*" alignment must be 4 in internal instruction RAM.

内蔵命令 RAM に配置した *section* セクションの整列条件は，4 の倍数を指定してください。

W4615: section "*section*" attribute is illegal in internal ROM/internal instruction RAM.

内蔵 ROM ／ RAM には，書き込み属性を持つ *section* セクションの配置は不適切です。

W4651: relocation entry in section "*section*" has unknown relocation type (*number*), ignored this entry.

セクション *section* 内のリロケーション・エントリが不正なリロケーション・タイプ *number* を持っています。このエントリは無視されます。

W4652: can not find *number* th symbol table entry for relocation of reference at offset(*offset*) in "*section*" section , this relocation is ignored.

セクション *section* のオフセット *offset* に存在する参照をリロケートするための *number* 番目のシンボル・テーブル・エントリが見つかりません。このリロケーションは無視されます。

W4653: relocation entry in relocation section "*section1*" used to relocate section "*section2*" has illegal r_offset(*offset*), ignored.

セクション *section2* のリロケーションに用いられるリロケーション情報セクション *section1* 内のエントリが不正なリロケーション・オフセット *offset* を持っています。このエントリは無視されます。

W4655: can not find GP-symbol in segment "*segment*" or illegal label reference for local symbol in file "*file2*"(section: *section2*, offset: *offset*, type: *relocatiion type*). local symbol is allocated in section "*section1*"(file: *file1*).

セグメント *segment* に GP シンボルが指定されていません，またはローカル・シンボルの配置／参照方法に矛盾があります。

[補足]

変数定義に対して #pragma section 指定で配置するセクションを変更した場合には,他のファイルにあるその変数に対する extern 宣言に対しても，同じ方法でセクションを変更してください。ローカル・シンボルは，ファイル *file1* の *section1* に配置されています。ローカル・シンボルに対する参照は，ファイル *file2* のセクション *section2* のオフセット *offset* に存在します。

W4656: can not find GP-symbol in segment "*segment*" or illegal label reference for symbol "*symbol*" in file "file2"(secion: *section2*, offset: *offset*, type: *relocation type*). "*symbol*" is allocated in section "*section1*"(file: file1).

セグメント *segment* に GP シンボルが指定されていません，またはシンボル *symbol* の配置／参照方法に矛盾があります。

[補足]

変数定義に対して #pragma section 指定で配置するセクションを変更した場合には,他のファイルにあるその変数に対する extern 宣言に対しても，同じ方法でセクションを変更してください。シンボル *symbol* は，ファイル *file1* の *section1* に配置されています。シンボル *symbol* に対する参照は，ファイル *file2* のセクション *section2* のオフセット *offset* に存在します。

W4657: can not find GP-symbol in section "*section*" of file "*file1*"or illegal label reference for symbol "*symbol*" in file "*file2*"(secion: *section2*, offset: *offset*, type: *relocation type*). "*symbol*" is allocated in section "*section1*"(file: *file1*).

ブート側実行形式ファイル *file1* のセクション *section* に GP シンボルが指定されていません，またはシンボル *symbol* の配置／参照方法に矛盾があります。

[補足]

変数定義に対して #pragma section 指定で配置するセクションを変更した場合には,他のファイルにあるその変数に対する extern 宣言に対しても，同じ方法でセクションを変更してください。シンボル *symbol* は，ファイル *file1* の *section1* に配置されています。シンボル *symbol* に対する参照は，ファイル *file2* のセクション *section2* のオフセット *offset* に存在します。

W4658: relocated value (*value*) of relocation entry (symbol: *symbol*, file: *file*, section: *section*, offset: *offset*, type: *relocation type*) for branch command become odd value.

分岐系のリロケーション・エントリ（シンボル *symbol*，ファイル *file*，セクション *section*，オフセット *offset*，リロケーション・タイプ *relocation type*）によってリロケートされた値 *value* が奇数になっています。

W4659: relocated value (*value*) of relocation entry (symbol: *symbol*, file: *file*, section: *section*, offset: *offset*, type: *relocation type*) for load/store command become odd value.

ロード／ストア系のリロケーション・エントリ（シンボル *symbol*，ファイル *file*，セクション *section*，オフセット *offset*，リロケーション・タイプ *relocation type*）によってリロケートされた値 *value* が奇数になっています。

W4661: symbol "*symbol*" (output section *section1*) is too far from output section "*section2*". (value: *value*, file: *file*, input section: *section3*, offset: *offset*, type: *relocation type*).

出力セクション *section2* から出力セクション *section1* に配置されたシンボル *symbol* への分岐命令が分岐可能範囲を越えています。分岐命令は，ファイル *file* 中のセクション *section3* のオフセット *offset* の位置に存在します。

W4662: output section "*section1*" is too far from output section "*section2*". (value: *value*, file: *file*, input section: *section3*, offset: *offset*, type: *relocation type*).

出力セクション *section2* から出力セクション *section1* に配置されたローカル・シンボルへの分岐命令が分岐可能範囲を越えています。分岐命令は，ファイル *file* 中のセクション *section3* のオフセット *offset* の位置に存在します。

W4663: output section "*section1*" overflowed or illeagal label reference for symbol "*symbol*" in file "*file2*" (value: value, input section: *section2*, offset: *offset*, type: *relocation type*). "*symbol*" is allocated in section "*section1*"(file: *file1*).

出力セクション *section1* がオーバーフローしました,またはシンボル *symbol* の配置されたセクションとラベル参照方法が不正です。*symbol* は，ファイル *file1* で *section1* に配置されています。不正な参照はファイル *file2* のセクション *section2* のオフセット *offset* に存在します。

W4664: output section "*section1*" overflowed or illeagal label reference for local symbol in file "*file2*" (value: value, input section: *section2*, offset: *offset*, type: *relocation type*). local symbol is allocated in section "*section1*"(file: *file1*).

出力セクション *section1* がオーバーフローしました，またはローカル・シンボルの配置されたセクションとラベル参照方法が不正です。ローカル・シンボルは，ファイル *file1* で *section1* に配置されています。不正な参照はファイル *file2* のセクション *section2* のオフセット *offset* に存在します。

W4702: no archive symbol table, ignored this archive file.

指定されたアーカイブ・ファイル内にアーカイブ・シンボル・テーブルが存在していません。
このアーカイブ・ファイルの指定は無視されます。

W4755: aligned odd value(*number1*) to be even value(*number2*).

奇数の値 *number1* を偶数の値 *number2* に整列しました。

W4757: segment "*segment*" overflowed highest address of target machine.

セグメント *segment* の割り付けが指示された領域がターゲット・マシンにおける可能なメモリ空間の範囲を越えています。

[補足]

-Ximem_overflow=warning **オプションを解除することにより，このメッセージをエラーにすることができます。**-rom_less **オプションを指定することにより，警告発生原因が内蔵** ROM **に関連する場合のみ，このメッセージを消去できます。**

W4769: *string* in segment directive is illegal when "-r" option specificated, ignored.

-r **オプション，または** -ro **オプションが指定された場合，セグメント・ディレクティブにおいて *string* を指定することはできません。無視されます。**

W4770: *string* in section directive is illegal when "-r" option specificated, ignored.

-r オプション，または -ro オプションが指定された場合，セクション・ディレクティブにおいて *string* を指定することはできません。無視されます。

W4772: no LOAD segments exist for mapping input section "*section*" in file "*file*", this section is mapped to non-LOAD *DUMMY* segment with no program header.

ファイル *file* 内のセクション *section* を割り付けることのできるセグメント・タイプ LOAD を持つセグメントが存在しません。このセクションは，プログラム・ヘッダを持たないロード可能でないダミーのセグメントに割り付けられます。

W4773: *string* is illegal in 1pass-mode, ignored, try in 2pass-mode ("-B" option).

1 パス・モードで *string* を指定することはできません。無視されます。-B オプションを用いた 2 パス・モードで指定してください。

W4774: *string* is illegal when "-f" option specified, ignored.

-f オプションが指定された場合，*string* を指定することはできません。無視されます。

W4775: *string* symbol multiply defined to segment "*segment*", first defined symbol "*symbol*" used.

セグメント *segment* に対する *string* シンボルが多重に定義されています。最初に定義されたシンボル *symbol* をセグメント *segment* に対する *string* シンボルとして用います。

W4777: *string* symbol multiply defined, first defined symbol "*symbol*" used.

string シンボルが多重に定義されています。最初に定義された *string* シンボル *symbol* を用います。

W4781: segment "*segment*" (*number1-number2*) must be in *string*-relative-address-able range (*number3-number4*).

セグメント *segment* の割り付けが指示された領域（*number1-number2*）が *string* 相対で参照可能な範囲（*number3-number4*）を越えています。

[補足]

このメッセージが出ると正しいアドレスを参照できない場合があります。セグメントを参照するときには，正しい割り付け位置になるようにしてください。

W4782: segment "*segment*" (*number1-number2*) overflowed highest or lowest address of internal memory (*number3-number4*).

セグメント *segment* の割り付けが指示された領域（*number1-number2*）がターゲット・マシンにおける可能なメモリ空間の範囲（*number3-number4*）を越えています。

[補足]

-Ximem_overflow=warning オプションを解除することにより，このメッセージをエラーにすることができます。-rom_less オプションを指定することにより，警告発生原因が内蔵 ROM に関連する場合のみ，このメッセージを消去できます。

W4783: *string* specified in EP symbol directive, ignored.

ep シンボル・ディレクティブにおいて *string* を指定することはできません。無視されます。

W4784: segment "*segment*" (*number1-number2*) overlaps guarded area (*number3-number4*).

セグメント *segment* の割り付けが指示された領域（*number1-number2*）がガード（使用禁止）（*number3-number4*）にオーバラップしています。

W4785: segment "*segment*" (*number1-number2*) overlaps programable peripheral I/O area (*number3-number4*).

セグメント *segment* の割り付けが指示された領域（*number1-number2*）がプログラマブル周辺 I/O 領域（*number3-number4*）にオーバラップしています。

W4788: section address specification is illegal when address of segment "*segment*" is not specified.

セグメント*segment*に対するアドレス指定がありません。セグメント内のセクションにアドレス指定を行った場合には，セグメントにもアドレス指定を行ってください。

W4852: "*string*" option needs argument, ignored.

***string* オプションは，引数を必要とします。無視されます。**

W4854: "*string*" option is ignored.

***string* オプションは無視されます。**

W4857: "*string1*" option is illegal when "*string2*" option is specified, ignored "*string1*" option.

***string2* オプションが指定された場合，*string1* オプションを指定することはできません。*string1* オプションは無視されます。**

W4860: "*string1*" option's argument is illegal, ignored "*string2*" option.

***string1* オプションに指定された引数が不正です。*string2* と指定されたオプションは無視されます。**

W4865: duplicated "*string*" option, ignored.

***string* オプションの指定が重複しています。無視されます。**

W4866: duplicated "*string1*" option, ignored "*string2*" option.

***string1* オプションの指定が重複しています。*string2* と指定されたオプションは無視されます。**

W4867: duplicated "*string1*" option, ignored "*string2*" option.

***string1* オプションの指定が重複しています。*string2* と指定されたオプションは無視されます。**

W4868: "*string*"option aligned odd value (*value1*) to be even value (*value2*).

***string* オプションに指定された奇数値 *value1* を偶数値 *value2* に整列しました。**

W4871: "*string*" option is not supported for V850 core.

***string* オプションは V850 コアのデバイスではサポートしていません。**

W4872: segment sort function is active, because new vector type exist in device file.

新しい割り込みタイプを持つデバイス・ファイルが指定されたため，セグメントのアドレス順ソートを実行します。このメッセージは，-Xolddir オプションにより CA850 Ver.2.50 以前の古いリンク・ディレクティブ規約を指定された場合に発生します。

W4873: "*string*" option is not supported for this device.

string オプションは，指定したデバイスにおいてサポートしていません。オプション指定は無視されました。

W4911: multiple inclusion of same file attempted, ignored.

同じファイルが入力ファイルとして複数回指定されています。

[注意事項]

PM+ を使用している場合，スタート・アップ・ファイルのアセンブリ言語ファイルをソース・ファイルとして登録し，かつスタート・アップ・ファイルのオブジェクト・ファイルをリンカ・オプションでスタート・アップ・ファイルとして指定している場合，この警告メッセージが出力されます。

この場合は，次のいずれかの対処を行ってください。

ソース・ファイルの登録から，スタート・アップ・ファイルを外し，[コンパイラ共通オプションの設定] ダイアログの “ スタートアップファイル ” にスタート・アップ・ファイルを指定する。

ダミーのアセンブリ言語ファイルを作成し，アセンブルして *.o ファイルを作成する。

そして，このファイルを [コンパイラ共通オプションの設定] ダイアログの “ スタートアップファイル ” に指定する。

B.1.5 ROM 化プロセッサ

F8400: b option needs argument.

-b オプションに対する引数が足りません。

F8401: o option needs argument.

-o オプションに対する引数が足りません。

F8402: p option needs argument.

-p オプションに対する引数が足りません。

F8403: t option needs argument.

-t オプションに対する引数が足りません。

F8404: F option needs argument.

-F オプションに対する引数が足りません。

F8405: unknown option argument.

オプションに対し指定することのできない引数が指定されています。

F8406: -*option* unknown option.

option オプションは，指定できません。

F8407: b option is specified more than once.

-b オプションが複数回指定されました。

F8411: *file* : illegal input file name.

入力ファイル *file* は，出力ファイル名と同じであるため入力できません。

F8412: illegal input file type. file(*file*) is archive file.

入力ファイル *file* は，アーカイブ・ファイルであるため入力できません。

F8413: *file* bad magic.

入力ファイル *file* は，不正であるため入力できません。

F8414: cannot open command file *file*.

コマンド・ファイル *file* が開けません。

F8415: nested command file *file*.

コマンド・ファイル *file* がネストしています。ネストはできません。

F8416: file name *name* is too long.

ファイル名 *name* の長さが限界を越えています。

F8417: cannot find device file.

デバイス・ファイルが見つかりません。

F8419: memory allocation fault.

メモリが足りません。

F8420: *file* : illegal section type "*section*" specified with -p option.

file 内の -p オプションで指定されたセクション *section* は，-p オプションでは指定できないセクション属性です。

F8421: *file* : illegal section type "*section*" specified with -t option.

file 内の -t オプションで指定されたセクション *section* は，-t オプションでは指定できないセクション属性です。

F8422: address of *symbol* symbol must be same in all files.

symbol のアドレスは，すべての入力ファイルで同じにしてください。

F8423: *file* : not absolute object.

リロケータブル・オブジェクト・ファイル *file* が入力ファイルとして指定されています。

F8424: *file* : "*symbol*l" symbol not found.

指定した *symbol* がオブジェクト・ファイル *file* 内に見つかりません。

F8425: rompsec section overflowed highest address of target machine.

rompsec セクションを作成する際にメモリの上限を越えました。

[備考]

-Ximem_overflow=warning オプションを指定することで，エラーを警告メッセージにすることが可能です。-rom_less オプションを指定することにより，本メッセージを消去することが可能です。

F8426: *section1* section and *section2* section overlapped.

section1 セクションと *section2* セクションが重なっています。

F8427: processor type must be same in all files.

不正な入力ファイルが指定されています。

F8428: symbol(*start_label*) must be word alignment.

start_label ラベルは，4 バイト境界のアドレスにしてください。

F8429: packing section not found.

指定したセクションがオブジェクト・ファイル内に見つかりません。

F8430: *section* section not found.

-p オプションで指定された *section* セクションが見つかりません。

F8432: illegal object file (*string*).

不正なオブジェクトファイルです。

F8453: can not open file *file*.

ファイル *file* をオープンできません。

W8508: duplicated -*option1* option, ignored -*option2* option.

option1 オプションが複数回指定されました。*option2* オプションは無視されます。

W8509: *section* section is already defined by -p option and therefore this section is ignored.

section セクションは，すでに -p オプションで指定されていますので無視されます。

W8510: *section* section is already defined by -t option and therefore this section is ignored.

section セクションは，すでに -t オプションで指定されていますので無視されます。

W8518: @ option needs argument, ignored.

@ オプションに対する引数が足りませんので無視されます。

W8525: rompsec section overflowed highest address of target machine.

rompsec セクションを作成する際にメモリの上限を越えました。

[備考]

-rom_less オプションを指定することにより，本メッセージを消去することが可能です。

W8533: *string* option needs argument, ignored.

string オプションに対する引数が足りません。オプション指定は無視されます。

W8534: *string* option's argument is illegal, ignored.

string オプションの引数が不正です。オプション指定は無視されます。

W8553: can not open file *file*

ファイル *file* を開けません。

W8554: "*option*" option's argument is illegal, ignored "*option*" option.

option オプションの引数が不正です。指定を無視しました。

B.1.6 ヘキサ・コンバータ

F8600: too many input files

複数の入力ファイルを指定することはできません。

F8601: too many output files

複数の出力ファイルを指定することはできません。

F8602: illegal option -character

-character **をオプションとして指定することはできません。**

F8603: expect format type [ITSs] after -f

-f **の後ろに** I，T，S，s **のいずれかを指定してください。**

F8604: expect section name after -I

-I **の後ろにセクション名を指定してください。**

F8605: expect block length after -b

-b **の後ろにブロック長を指定してください。**

F8606: expect disp value after -d

-d **の後ろにオフセット値を指定してください。**

F8607: expect input file

入力ファイル名を指定してください。

F8608: expect output file after -o

-o **の後ろに出力ファイル名を指定してください。**

F8609: expect device file path after -F

-F **の後ろにデバイス・ファイル・パスを指定してください。**

F8610: illegal use of *option* option

option **オプションの指定方法が不正です。**

F8611: nested command file *file*

コマンド・ファイル *file* **がネストしています。ネストはできません。**

F8612: no section data exists in specified address area (*address1-address2*)

-U **オプションで指定された領域（***address1-address2***）にセクションがありません。**

F8613: file name *name* is too long

ファイル名 *name* **の長さが限界を越えています。**

F8620: cannot open file *file*

ファイル *file* **オープンできません。**

F8621: cannot open output file *file*

出力ファイル *file* **をオープンできません。**

F8622: cannot get section *section*

section セクションが見つかりません。

F8623: cannot find device file

デバイス・ファイルが見つかりません。

F8624: cannot find device information

デバイスの情報が見つかりません。

F8625: *file* is not ELF file

ファイル *file* は，ELF 形式のオブジェクト・ファイルではありません。

F8626: *file* is archive file

ファイル *file* は，アーカイブ・ファイルです。

アーカイブ・ファイルを指定することはできません。

F8627: illegal target machine type

ターゲット・マシンのタイプが不正です。

F8628: illegal object file (*string*)

オブジェクトファイルが不正です。

F8629: cannot create HEX rom data, because there is no memory information

メモリ情報がないため，ROM データを作成できません。

F8630: *section* section overflowed lowest address of internal memory

section セクションは，内蔵 ROM 領域，または -U オプションで指定した領域の下限を越えています。

F8639: *section* : no such section

指定されたセクション *section* が見つかりません。

F8640: illegal block length *length*

-b オプションで指定されたブロック長の値 length が不正です。

F8641: illegal disp value *value*

-d オプションで指定されたオフセットの値 *value* が不正です。

F8642: illegal fill value

-U オプションで指定された充填値の値が不正です。

F8643: illegal start address *value*

-U オプションで指定されたスタートアドレスの値 *value* が不正です。

F8644: illegal size value *value*

-U オプションで指定されたサイズの値 *value* が不正です。

F8645: size must not be 0

-U オプションで指定するサイズは 0 にはできません。

F8646: memory allocation fault

メモリが足りません。

F8651: specified address area(*addr1* - *addr2*) overlaps I/O area (*addr3* - *addr4*)

-U オプションで指定した範囲（*addr1- addr2*）が周辺 I/O（*addr3 - addr4*）と重なっています。

W8713: file name *name* is too long

ファイル名 *name* の長さが限界を越えています。

W8714: expect command file after @, ignored

@ の後ろにコマンド・ファイルを指定してください。@ オプションは無視されます。

W8715: *section* section is already defined by -I option, ignored

***section* セクションは，-I オプションで指定済みなので無視されます。**

W8716: -S and -x expect -fT

-S オプションと -x オプションは，拡張テック・ヘキサ・フォーマットが指定された場合にのみ有効です。

W8717: -S expect -fT

-S オプションは，拡張テック・ヘキサ・フォーマットが指定されたが指定された場合にのみ有効です。

W8718: -x expect -S and -fT

-x オプションは，-S オプションと -fT オプションと同時に指定された場合にのみ有効です。

W8719: *option1* option overrides *option2* option

***option1* オプションが指定されたため，*option2* オプションは無効となります。**

W8730: *section* section overflowed lowest address of internal memory

***section* セクションは，内蔵 ROM 領域，または -U オプションで指定した領域の下限を越えています。**

W8731: *section* section overflowed highest address of internal memory

***section* セクションは，内蔵メモリ空間の範囲を越えています。**

W8732: *section* section overflowed lowest address of program memory

***section* セクションは，プログラムメモリの下限を越えています。**

W8733: *section* section is converted from its midst

***section* セクションの途中の指定されたアドレスからヘキサ変換を行います。**

W8734: *section* section is converted until its midst

***section* セクションの途中の指定された領域までヘキサ変換を行います。**

W8735: The address of convert area exceeds the maximum value of the address that can be expressed in the Intel expanded hex format

アドレスが，インテル拡張ヘキサ・フォーマット形式で表現可能なアドレスの最大値（20 ビット）を越えています。

表現可能な範囲をアドレスとして出力する形で処理を続行します。

[補足]

本メッセージの原因としては，次の理由が考えられます。

(1) **ROM 化を忘れている**

ROM に配置したセクションと同時に内蔵 RAM のセクションをヘキサ変換を行おうとしている可能性があります。ヘキサ変換を行う前に ROM 化を行ってください。

(2) **ヘキサ変換対象のセクションが間違っている**

同時にヘキサ変換するセクションのアドレスが，大きく離れている可能性があります。複数の ROM を使用している場合には，ROM ごとにヘキサ変換を行ってください。また，ヘキサ変換するセクションは正しいか，セクションの配置アドレスが正しいかを確認してください。

(3) **セクションのサイズ自体が大きい**

ヘキサ変換を行おうとしている領域が大きく表現可能な範囲を越えている場合には，ヘキサ変換できません。表現可能な範囲まで分割するか，別のヘキサ・フォーマットを使用してください。

[注意事項]

アドレスの最大値を 20 ビットと表現していますが，実際には 20 ビット付近となります。本メッセージが出力される原因であるインテル拡張ヘキサ・フォーマット形式の 20 ビットのアドレス表現は，上位アドレスを保有する拡張アドレス・レコードと，そこからのオフセットを所有するデータ・レコードから算出されます。拡張アドレス・レコードの上位アドレスが 20 ビットに収まっていれば，それに続くデータ・レコードのオフセットを加算した結果が 20 ビットを越えたとしても，インテル拡張ヘキサ・フォーマット形式として正常な出力であり，エラーにはなりません。

W8736: The address of convert area exceeds the maximum value of the address that can be expressed in the Motorola S type hex format (standard address)

アドレスがモトローラ S タイプ・ヘキサ・フォーマット（スタンダード・アドレス）形式で表現可能なアドレスの最大値（24 ビット）を越えています。

表現可能な範囲をアドレスとして出力する形で処理を続行します。

[補足]

本メッセージの原因としては，次の理由が考えられます。

(1) **ROM 化を忘れている**

ROM に配置したセクションと同時に内蔵 RAM のセクションをヘキサ変換を行おうとしている可能性があります。ヘキサ変換を行う前に ROM 化を行ってください。

(2) **ヘキサ変換対象のセクションが間違っている**

同時にヘキサ変換するセクションのアドレスが，大きく離れている可能性があります。複数の ROM を使用している場合には，ROM ごとにヘキサ変換を行ってください。また，ヘキサ変換するセクションは正しいか，セクションの配置アドレスが正しいかを確認してください。

(3) **セクションのサイズ自体が大きい**

ヘキサ変換を行おうとしている領域が大きく表現可能な範囲を越えている場合には，ヘキサ変換できません。表現可能な範囲まで分割するか，別のヘキサ・フォーマットを使用してください。

W8737: The start address of convert area exceeds the maximum value of the address that can be expressed in the Intel expanded hex format

先頭アドレスが，インテル拡張ヘキサ・フォーマット形式で表現可能なアドレスの最大値（20 ビット）を越えています。表現可能な範囲をアドレスとして出力する形で処理を続行します。

[補足]

本メッセージの原因としては，次の理由が考えられます。

(1) **セクションのアドレスが大きい**

セクションの配置されているアドレスが，表現可能な範囲を越えている可能性があります。-d オプションでヘキサ変換する領域の先頭アドレスを指定してそのアドレスからのオフセットとしてください。

(2) **-d オプションで指定した値が不適切**

-dオプションを指定することで,アドレスを指定した値からのオフセットとして扱うことが可能です。この値からのオフセットが表現可能な範囲を越えている可能性があります。適切な値を指定してください。

W8738: The start address of convert area exceeds the maximum value of the address that can be expressed in the Motorola S type hex format (standard address)

先頭アドレスが，モトローラ S タイプ・ヘキサ・フォーマット形式で表現可能なアドレスの最大値（24ビット）を越えています。表現可能な範囲をアドレスとして出力する形で処理を続行します。

[補足]

本メッセージの原因としては，次の理由が考えられます。

(1) セクションのアドレスが大きい

セクションの配置されているアドレスが，表現可能な範囲を越えている可能性があります。-d オプションでヘキサ変換する領域の先頭アドレスを指定してそのアドレスからのオフセットとしてください。

(2) -d オプションで指定した値が不適切

-dオプションを指定することで,アドレスを指定した値からのオフセットとして扱うことが可能です。この値からのオフセットが表現可能な範囲を越えている可能性があります。適切な値を指定してください。

W8747: too small block length. Length => *length*

指定されたブロック長の最大値が小さすぎます。デフォルトの値 *length* に変更し処理を続行します。

W8748: too large block length. Length => *length*

指定されたブロック長の最大値が大きすぎます。指定することのできる値の最大値 *length* に変更し処理を続行します。

W8749: block length is set. Length => *length*

ブロック長の最大値をデフォルトの値から *length* に変更し処理を続行します -b オプション時に指定することのできる値が指定された場合に出力されます。

W8750: symbol block length exceed default value

シンボル・ブロックのブロック長が指定されたブロック長の最大値を越えています。

B.1.7 アーカイバ

F8200: memory allocation fault

メモリが足りません。

F8201: bad key *character* -- use [dm(a|b)qr(a|b|u)txV]

***character* をキーとして指定することはできません。**

F8202: bad option *character* -- use [cv]

***character* をオプションとして指定することはできません。**

F8203: bad option *string*

***string* をオプションとして指定することはできません。**

F8204: can not create file *file*

ファイル *file* を作成できません。

F8205: file name *name*... is too long

ファイル名 *name* の長さが限界を越えています。

F8206: can not open file *file*

ファイル *file* をオープンできません。

F8207: can not close file *file*

ファイル *file* をクローズできません。

F8208: can not read file *file*

ファイル *file* からの読み込みができません。

F8209: can not write file *file*

ファイル *file* への読み込みができません。

F8210: can not seek file *file*

ファイル *file* をシークできません。

F8212: can not nest command file *file*

コマンド・ファイル *file* がネストしています。ネストはできません。

F8213: *file* is not archive file

ファイル *file* は，アーカイブ・ファイルではありません。

F8214: malformed archive file *file*

アーカイブ・ファイル *file* の内容が破壊されているおそれがあります。

F8215: can not find member *member*

アーカイブ・ファイル内にメンバ *member* が存在しません。

F8216: symbol table limit error *file* (*number1*) -- limit is *number2*

アーカイブ･ファイル *file* において,シンボルの個数 *number1* が限界を越えました｡限界値は *number2* です。

F8217: symbol table error *file*

アーカイブ・ファイル *file* において，アーカイブ・ストリング・テーブルの内容が破壊されているおそれがあります。

F8218: string table error *file*

アーカイブ・ファイル *file* において，アーカイブ・シンボル・テーブルの作成に失敗しました。

F8219: *file* has no member

アーカイブ・ファイル *file* 内にメンバが存在しません。

F8220: version error *file*

指定されたファイル*file*の形式のバージョンがこのアーカイバの扱うことのできるバージョンではありません。

F8221: can not read archive header *file*

アーカイブ・ファイル *file* のヘッダの読み込みができません。

W8306: can not open file *file*

ファイル *file* をオープンできません。

W8307: can not close file *file*

ファイル *file* をクローズできません。

W8308: can not read file *file*

ファイル *file* から読み込みできません。

W8309: can not write file *file*

ファイル *file* への書き込みができません。

W8310: can not seek file *file*

ファイル *file* をシークできません。

W8311: can not find file *file*

ファイル *file* の読み込みができません。

W8315: can not find member *member*

アーカイブ・ファイル内にメンバ *member* が存在しません。

W8322: this symbol offset not true

アーカイブ・ファイルのシンボル・オフセットが不正です。

B.1.8 セクション・ファイル・ジェネレータ

F8000: cannot open output file *file*

出力ファイル *file* を作成できません。

F8010: cannot open input file *file*

入力ファイル *file* をオープンできません。

F8020: cannot write file ' *file*'

出力ファイル *file* に書き込みできません。

F8030: unknown option *option*

sf850 にはないオプション *option* が指定されました。

F8040: -cl level out of range (0 - 2)

-cl オプションで指定された数値が間違っています。

F8050: *option* option need sub argument

オプション *option* には引数が必要です。

F8080: not enough memory

メモリが足りません。

W8101: cannot calculate *name*'s frequency

変数 *name* の利用頻度が計算できません。変数が利用されていない場合で，かつ，-v オプション指定時に出力されます。この警告メッセージは無視してかまいません。

W8111: you use -O option, sorting option ignored

-O オプションと，ソート・オプションが同時に指定されたため，ソート・オプションは無視されます。

W8121: unrecognized option *option*, ignored

オプション *option* が認識できないため，無視されます。

B.1.9 ダンプ・コマンド

F9001: can not open file *file*

ファイル *file* をオープンできません。

F9003: nested command file *file*

コマンド・ファイル *file* がネストしています。ネストできません。

F9024: memory allocation error

メモリが足りません。

W9102: illegal object file (*string*)

オブジェクト・ファイルが正しくありません。

W9121: buffer number error

バッファナンバーが正しくありません。

W9122: illegal option -c

-c をオプションとして，指定することはできません。

W9123: illegal option +c

+c をオプションとして，指定することはできません。

W9125: not enough argument

引数が足りません。

W9126: not enough argument for *string*

オプション *string* に対する引数が足りません。

W9127: Size error

サイズが正しくありません。

B.1.10 ディスアセンブラ

F8801: bad magic file *file*

指定した *file* が V850 **マイクロコントローラのオブジェクト・ファイルではありません。**

F8802: cannot find device file

デバイス・ファイルが見つかりません。

F8803: cannot open file *file*

ファイル *file* をオープンすることができません。

F8804: illegal object (*string*)

オブジェクト・ファイルが正しくありません。

F8805: nested command file *file*

コマンド・ファイル *file* がネストしています。

ネストはできません。

F8821: memory allocation error

メモリが足りません。

B.1.11 クロス・リファレンス・ツール

F9600: %s

エラーを検出しました。

F9601: '%s' can't open

ファイルのオープンに失敗しました。

F9602: '%s' can't seek

ファイルのシークに失敗しました。

F9603: '%s' can't read

ファイルの読み込みに失敗しました。

F9604: '%s' can't write

ファイルの書き込みに失敗しました。

F9605: no memory

メモリの確保に失敗しました。

F9606: '%s' not found

プリプロセッサがファイル %s を検出できません。

F9607: '%s' failed

プリプロセッサがファイル %s でエラーを検出しました。

F9608: input file nothing

入力ファイルが指定されていません。

F9609: '%s' not specified file name

オプション %s でファイル名の指定が行われていません。

F9610: '%s' not specified identifier

オプション %s で識別子の指定が行われていません。

F9611: '%s' not specified symbol

オプション %s でシンボルの指定が行われていません。

F9612: '%s' range over

オプション %s に範囲外の値を指定しました。

F9613: '%s' not specified path

オプション %s にパスの指定が行われていません。

F9614: multiple declaration function '%s'

関数名 %s が重複しています。

W9651: '%s' invalid option, ignored

無効なオプション %s を指定しました。

W9652: too long identifier '%s...' [%d]

識別子 %s が %d 文字を越えました。

W9653: specified function '%s' not found

指定した関数 %s に関する記述がありません。

W9654: section not found in '%s'

-file=%s で指定したファイル %s にセクションに関する記述がありません。

W9655: DefinitionType section not found in '%s'

-d=%s で指定したファイル %s に DefinitionType セクションに関する記述がありません。

W9656: Ignoreldent section not found in '%s'

-i=%s で指定したファイル %s に Ignoreldent セクションに関する記述がありません。

W9657: NoIncludeFile section not found in '%s'

-ni=%s で指定したファイル %s に NotIncludeFile セクションに関する記述がありません。

C9690: %s

コンパイル・エラー %s が発生しました。

C9691: null pointer access

NULL ポインタにアクセスしようとしました。

C9692: index out of range

配列の範囲外にアクセスしようとしました。

B.1.12 メモリ・レイアウト視覚化ツール

F9700: %s

エラーを検出しました。

F9701: '%s' can't open

ファイルのオープンに失敗しました。

F9702: '%s' can't seek

ファイルのシークに失敗しました。

F9703: '%s' can't read

ファイルの読み込みに失敗しました。

F9704: '%s' can't write

ファイルの書き込みに失敗しました。

F9705: no memory

メモリの確保に失敗しました。

F9706: '%s' is not ELF executable file

ファイル %s **が** ELF **実行形式ではありません。**

F9707: input file nothing

入力ファイルが指定されていません。

F9708: too many files

解析対象ファイルを複数指定しています。

F9709: Executable file does not provide symbol information

解析対象ファイルにシンボル・テーブルが存在しません。

F9710: Executable file does not provide data-object information

解析対象ファイルに変数が 1 **個も存在しません。**

F9711: '%s' not found range

-r **オプションで範囲指定が行われていません。**

F9712: '%s' invalid range

-r **オプションで範囲外の値を指定しました。**

F9713: '%s' invalid end-address

-r **オプションで指定した末尾アドレスが不正です。**

F9714: '%s' not spevified path

-r **オプションで出力パス指定が行われていません。**

F9715: '%s' not specified file name

-r **オプションで出力ファイル指定が行われていません。**

W9751: '%s' invalid option, ignored

無効なオプション %s を指定しました。

C9790: %s

コンパイル・エラー %s が発生しました。

C9791: null pointer access

NULL ポインタにアクセスしようとしました。

C9792: index out of range

配列の範囲外にアクセスしようとしました。

B.2 PM+ から起動時のメッセージ

この節では，PM+ から起動時のメッセージ・ダイアログについて説明します。

B.2.1 メッセージの形式

PM+ から起動時に出力するメッセージは，図 B － 1 に示すメッセージ・ダイアログ内に表示されます。

図 B － 1 メッセージ・ダイアログの例

B.2.2 コンパイラ共通

BPC レジスタの指定が不正です。

BPC レジスタに整数以外の文字を使用しないでください。または，デバイスで定められた範囲外です。

rompcrt ファイルの指定が不正です。

rompcrt ファイルのファイル名に空白を使用しないでください。または，ソース・ファイル・リストの中に rompcrt ファイルのソース・ファイルと拡張子を除いたファイル名が同じファイルを使用しないでください。

最終出力ディレクトリは存在しません。作成しますか？

最終出力用フォルダが存在しないので作成するかどうか確認してください。

スタートアップファイルの指定が不正です。

スタート・アップ・ファイルのファイル名に空白を使用しないでください。または，ソース・ファイル・リストの中にスタート・アップ・ファイルのソース・ファイルと拡張子を除いたファイル名が同じファイルを使用しないでください。

セキュリティ ID の指定が不正です。

セキュリティ ID に 16 進数以外の文字を使用しないでください。または，セキュリティ ID が定められた範囲外です。

中間出力ディレクトリは存在しません。作成しますか？

中間出力用フォルダが存在しないので作成するかどうか確認してください。

ディレクトリを作成できません。

フォルダが作成できません。

ブートオブジェクトファイルの指定が不正です。

ブート・オブジェクト・ファイルのファイル名に空白を使用しないでください。

ファイルの拡張子が存在しません。

ビルドが正常に動作しない可能性があります。ファイル名には拡張子が必要です。拡張子がないと，ビルド処理が正常に動作しない場合があります。

分岐テーブルのアドレスの指定が不正です。

分岐テーブルのアドレスに 16 進数以外の文字を使用しないでください。

リンクディレクティブファイルの指定が不正です。

リンク・ディレクティブ・ファイルのファイル名に空白を使用しないでください。

B.2.3 コンパイラ

Far Jump ファイルの指定が不正です。

Far Jump ファイルのファイル名に空白を使用しないでください。

sconst に配置するデータ長の指定が不正です。

sconst に配置するデータ長に数字以外の文字を使用しないでください。

sdata/sbss に配置するデータ長の指定が不正です。

sdata ／ sbss に配置するデータ長に数字以外の文字を使用しないでください。

アセンブラソースのディレクトリは存在しません。作成しますか？

アセンブリ言語ソースのフォルダが存在しないので作成するかどうか確認してください。

アセンブルリストのディレクトリは存在しません。作成しますか？

アセンブル・リストのフォルダが存在しないので作成するかどうか確認してください。

インライン展開するコードサイズの上限の指定が不正です。

インライン展開するコード・サイズの上限に数字以外の文字を使用しないでください。

インライン展開するスタックサイズの上限の指定が不正です。

インライン展開するスタック・サイズの上限に数字以外の文字を使用しないでください。

エラー数の上限の指定が不正です。

エラー数の上限に数字以外の文字を使用しないでください。

オプションの変更内容を全てのソースに反映しますか？

ソース個別オプションの変更内容をすべてのソースに反映するかどうか確認してください。

外部変数のソートの指定が不正です。

外部変数のソートのファイル名には，拡張子を除いたファイル名が同じ，拡張子に不正な文字，およびコンマを使用しないでください。

セクションファイルの指定が不正です。

セクション・ファイルのファイル名に空白を使用しないでください。

セミコロンは使用できません。

複数項目入力可能なオプションは “ ; （セミコロン）” を使用しないでください。

ディレクトリを作成できません。

フォルダを作成できません。

テンポラリディレクトリは存在しません。作成しますか？

テンポラリ・フォルダが存在しないので作成するかどうか確認してください。

頻度情報ファイルのディレクトリは存在しません。作成しますか？

頻度情報ファイルのフォルダが存在しないので作成するかどうか確認してください。

ファイルの拡張子が存在しません。ビルドが正常に動作しない可能性があります。

ファイル名には拡張子が必要です。拡張子がないと，ビルド処理が正常に動作しない場合があります。

マクロの上限数の指定が不正です。

マクロの上限数に数字以外の文字を使用しないでください。

ループ展開数の指定が不正です。

ループ展開数に数字以外の文字を使用しないでください。

B.2.4 アセンブラ

`sdata/sbss` **に配置するデータ長の指定が不正です。**

sdata ／ sbss **に配置するデータ長に数字以外の文字を使用しないでください。**

オプションの変更内容を全てのソースに反映しますか？

ソース個別オプションの変更内容をすべてのソースに反映するかどうか確認してください。

セミコロンは使用できません。

複数項目入力可能なオプションは“;（セミコロン）”を使用しないでください。

B.2.5 リンカ

`rompcrt` **ファイルの指定が不正です。**

ソース・ファイル・リストの中に rompcrt **ファイルのソース・ファイルと拡張子を除いたファイル名が同じファイルを使用しないでください。**rompcrt **ファイルの指定はデフォルトになります。**

このプロジェクトは `rompcrt` **ファイルは必要ありません。**

ROM **化を指定していないプロジェクトでは** rompcrt **ファイルは必要ありません。**

このプロジェクトはブートオブジェクトファイルは必要ありません。

フラッシュ対応オブジェクトの指定をしていないプロジェクトではブート・オブジェクト・ファイルは必要ありません。

出力ファイルの指定が不正です。

出力ファイル名には，romp.out**，拡張子に不正な文字，および空白を使用しないでください。**

スタートアップファイルの指定が不正です。

ソース・ファイル・リストの中にスタート・アップ・ファイルのソース・ファイルと拡張子を除いたファイル名が同じファイルを使用しないでください。スタート・アップ・ファイルの指定はデフォルトになります。

セミコロンは使用できません。

複数項目入力可能なオプションは“;（セミコロン）”を使用しないでください。

選択したファイルはこのプロジェクトから削除できません。

スタートアップファイル，および rompcrt ファイルはプロジェクトから削除できません。

ホールのフィリング値の指定が不正です。ホールのフィリング値に 16 進以外の文字を使用しないでください。

B.2.6 ROM 化プロセッサ

ROM 化セクションファイルに同じマクロ名が存在します。

ROM 化セクションファイルに同じマクロ名は使用しないでください。正常に出力されません。

ROM 化セクションファイルの出力は失敗しました。

ROM 化セクションファイルは正常に出力されません。

ROM 化セクションファイルの出力は成功しました。

ROM 化セクションファイルは正常に出力されます。

出力ファイルの指定が不正です。

出力ファイル名には，a.out，拡張子に不正な文字，および空白を使用しないでください。

セミコロンは使用できません。

複数項目入力可能なオプションは“;（セミコロン）”を使用しないでください。

B.2.7 ヘキサ・コンバータ

ROM 領域のサイズの指定が不正です。

ROM 領域のサイズに 16 進数の数字以外の文字や，範囲外の値を指定しないでください。

ROM 領域のスタートアドレスの指定が不正です。

ROM 領域の先頭アドレスに 16 進数の数字以外の文字や，範囲外の値を指定しないでください。

ROM 領域のフィリング値の指定が不正です。

ROM 領域のフィリング値に 16 進数の数字以外の文字や，範囲外の値を指定しないでください。

出力するアドレスのオフセットの指定が不正です。

出力するアドレスのオフセットに 16 進数と 10 進数の数字以外の文字を使用しないでください。

出力ファイルの指定が不正です。

出力ファイル名には，拡張子に不正な文字，および空白を使用しないでください。ブロック／レコードの最大長の指定が不正です。ブロック／レコードの最大長に 10 進数の数字以外の文字や，範囲外の値を指定しないでください。

セミコロンは使用できません。

複数項目入力可能なオプションは “ ; （セミコロン） ” を使用しないでください。

B.2.8 アーカイバ

アーカイブファイルの指定が不正です。

アーカイブ・ファイル名には，拡張子に不正な文字，および空白を使用しないでください。

B.2.9 セクション・ファイル・ジェネレータ

tidata セクションの割り付け可能サイズの指定が不正です。

tidata セクションの割り付け可能サイズに 10 進数以外の文字を使用しないでください。

tidata.byte セクションの割り付け可能サイズの指定が不正です。

tidata.byte セクションの割り付け可能サイズに 10 進数以外の文字を使用しないでください。

出力ファイルの指定が不正です。

出力ファイル名には，拡張子に不正な文字，および空白を使用しないでください。

デバイスファイルを正常にオープンできません。

デバイス・ファイルを正常にオープンできません。

ファイルの拡張子が存在しません。

ファイル名には拡張子が必要です。拡張子がないと，ビルド処理が正常に動作しない場合があります。

B.2.10 クロス・リファレンス・ツール，およびメモリ・レイアウト視覚化ツール

F102: 指定したファイルの読み込みに失敗しました。

指定したファイルの読み込みに失敗したため，エラーが発生したファイル名を表示します。指定したファイルが壊れていないかを確認してください。

F103: 指定したファイルは，デバイスファイルではありません。正しいファイル名を指定したか確認してください。

指定したプロジェクト・ファイルに関連したデバイス・ファイルを読み込みましたが，指定しているファイルはデバイス・ファイルではありませんでした。そのため，エラーが発生したファイル名を表示します。正しいプロジェクト・ファイルを指定してください。

F105: NECDEV レジストリ中にデバイス情報が見つかりませんでした。デバイス・ファイルがインストールされているか確認してください。

指定したプロジェクト・ファイルに対応したデバイス情報がレジストリ中に見つかりませんでした。使用するデバイス・ファイルをインストールしてください。

F106：指定したフォルダは，存在しません。作成しません。作成しますか？

指定したフォルダが存在しません。

F107：指定したフォルダは，見つかりません。正しいフォルダ名を指定したか確認してください。

指定したフォルダが見つかりません。フォルダ名を確認して，正しいフォルダを指定してください。

F108：指定したフォルダの作成に失敗しました。正しいフォルダ名を指定したか確認してください。

指定したフォルダの作成に失敗しました。指定したフォルダを調べて，正しいフォルダを指定してください。

F109：指定したファイルを開くことは，できませんでした。ファイルを開くアプリケーションをエクスプローラで選択してください。

解析結果を表示しようとしましたが，プログラムの関連付けがされていないためにプログラムの実行に失敗しました。プログラムの関連付けをエクスプローラで行ってください。

F110：指定したファイルは，ReadOnly ファイルです。別のファイル名で保存してください。

保存時に指定したプロジェクトファイルが ReadOnly 属性でした。別の名前で保存するか，ReadOnly 属性を解除後，保存を行ってください。

F120：解析対象ファイルが指定されていません。

解析対象ファイルが指定されていません。解析対象ファイルを指定してください。

F121：出力ファイルが指定されていません。詳細オプション設定で出力するファイル形式とファイル名を指定してください。

解析結果を出力するファイルの種類，ファイル名が指定されていません。詳細オプション設定で出力ファイルの設定を行ってください。

F122：xxxx が不正です。yyyy までの値を入力してください。

指定した数値が範囲外です，または記号を入力しています。xxxx には，不正な値を設定しているエディット・ボックスの名前を，yyyy には入力可能な数値範囲を表示します。正しい値を入力してください。

F123：開始アドレスが不正です。0 から 0xffffffff までの値を入力してください。

指定した開始アドレスが正しくありません。正しい値を入力してください。

F124：終了アドレスが不正です。0 から 0xffffffff までの値を入力してくだい。

指定した終了アドレスが正しくありません。正しい値を入力してください。

F125：アドレス範囲が無効です。

指定したアドレス範囲が正しくありません。正しい値を入力してください。

F126：解析結果に失敗しました。ログファイルを開きますか？

解析に失敗しました。ログファイルを開いてエラーの原因を参照したあとに，適切な処理を行い再度解析を行ってください。

F128：関数名読み込み用のファイル名を指定してください。

解析に失敗しました。関数名読み込み用のファイルを指定していません。ファイルを指定してください。

F129：**出力するファイル名を指定してください。**

出力ファイル名を設定していません。ファイルを指定してください。

F130：**出力範囲が指定されていません。全範囲出力を設定します。よろしいですか？**

RAM **マップの出力範囲を指定していません。**RAM **マップの出力範囲を設定するか，全範囲を出力するを選択してください。**

F131：**識別子名読み込み用のファイル名を指定してください。**

クロス・リファレンスのオプション設定において，識別子名読み込み用のファイルを指定していません。ファイルを指定してください。

B.3 stk850 のメッセージ

この節では，stk850 のメッセージについて説明します。

B.3.1 メッセージの形式

stk850 のメッセージは，次の形式でメッセージ表示部に表示されます。

表 B － 1 stk850 のメッセージ形式

番号	意味
E93xx	エラー・メッセージ
W940x	ファイル全般の警告メッセージ
W941x，W942x	スタック・サイズ指定ファイル関連の警告メッセージ
W943x	中間アセンブリ言語ファイル関連の警告メッセージ
W945x	出力ファイル関連の警告メッセージ
W946x	スタック・サイズ指定関連の警告メッセージ
I95xx	確認メッセージ

B.3.2 メッセージ

E9300：プロジェクト・ファイル（*path*）が見つかりませんでした。

プロジェクト・ファイルが存在しません。プロジェクト・ファイルが存在するか確認してください。

E9301：（*path*）は読み込み禁止です。ファイルを読み込める状態にしてください。

プロジェクト・ファイルか，stk システム・ファイルが読み込み禁止，または *path* にフォルダを指定した場合に発生します。

E9302：（*path*）の（*line*:*string*）は，不正なフォーマットです。ファイルを作り直して下さい。

ファイルの *line*:*string* は，不正なフォーマットです。プロジェクト・ファイルか，stk システム・ファイルで不正なフォーマットを発見した場合に発生します。

E9303：（*path*）の起動に失敗しました。インストールし直してください。

stk850 の起動に失敗しました。DLL から，exe の起動に失敗した場合に発生します。インストールが正常に行われていないか，exe ファイルが壊れています。再インストールが必要です。

W9400：ファイル（*path*）が見つかりませんでした。ファイル名を指定し直してください。

指定されたファイルが見つかりません。正しいファイル名を指定してください。

W9401：ファイル（*path*）は読み込み禁止です。ファイルを読み込める状態にしてください。

ファイルは読み込み禁止です。ファイルを読み込める状態にしてください。

W9402: ファイル（*path*）は読み込み専用です。ファイルもしくはディレクトリに書き込み許可属性を設定して下さい。

ファイルは読み込み専用です。ファイル，またはフォルダに書き込み許可属性を設定してください。

W9403: ファイル（*path*）の読み込みでエラーが発生しました。ファイルが読み込み可能な状態にあるか確認してください。

ファイルの読み込みでエラーが発生しました。ファイルが読み込み可能な状態にあるか確認してください。

W9404: ファイル（*path*）への書き込みでエラーが発生しました。ファイルが書き込み可能な状態にあるか確認してください。

ファイルの書き込みでエラーが発生しました。ファイルが書き込み可能な状態にあるか確認してください。

W9405: ファイル（*path*:*line*）は一行の文字数限界（*num*）を越えています。一行を短くしてください。

ファイル（*path*:*line*）は一行の文字数限界（*num*）を越えています。各ファイルの限界値は，「14.5.1 stk850 の限界値」を参照してください。

W9406: ファイル（*path*:*line*）は一行の登録限界数（*num*）を越えています。登録情報の数を減らしてください。

ファイル（*path*:*line*）は一行の登録限界数（*num*）を越えています。登録情報の数を減らしてください。各ファイルの限界値は，「14.5.1 stk850 の限界値」を参照してください。

W9407: ファイル名（*path*）が長すぎます。255 文字以上となるファイルを扱うことはできません。

ファイル名（*path*）が長すぎます。255 文字以上となるファイルを扱うことはできません。

W9408: 指定されたファイル（*path*:*line*）は，（*type*）として正しくありません。正しいファイルを指定してください。

指定されたファイルは，*type* として正しくありません。正しいファイルを指定してください。起動後にメニューで指定したプロジェクト・ファイルが不正な場合，stk システム・ファイルが不正な場合，スタック・サイズ指定ファイルとして不正なファイルを指定した場合に発生します。

W9410: スタックサイズ指定ファイル（*path*:*line*）に，宣言されていない関数名（*function*）が指定されています。

スタック・サイズ指定ファイルに，プロジェクトで使われていない関数（*function*）に対する設定が見つかました。関数名の確認をしてください。[読み込みを中止しますか？] ダイアログが出力されます。

表 B － 2 [読み込みを中止しますか？] ダイアログ

ボタン	内容
中止	読み込みを中止します。
再試行	エラーの行を無視し，次の行から読み込みを継続します。
無視	エラーの行を無視し，次の行から読み込みを継続します。以降の警告（W9410 ～ W9426）に対して，このダイアログを抑止しますが，メッセージ表示部への出力は行います。

W9411: スタックサイズ指定ファイル（*path*:*line*）で，スタティック関数を宣言したファイル名が長すぎます。255 文字以下にしてください。

スタック・サイズ指定ファイルで，スタティック関数を宣言したファイル名が長すぎるものが見つかりました。255 文字以下にしてください。[読み込みを中止しますか？] ダイアログが出力されます。

W9412: スタックサイズ指定ファイル（*path*:*line*）で，呼び出し関数に長すぎる関数名が見つかりました。1,022 文字以下にしてください。

スタック・サイズ指定ファイルで，呼び出し関数の指定に長すぎる関数名が見つかりました。1,022 文字以下にしてください。[読み込みを中止しますか？] ダイアログが出力されます。

W9413: スタックサイズ指定ファイル（*path*:*line*）で，不正な加算情報がありました（*value*）。加算サイズは，“ ADD= ” の後に，10 進数または，“ 0x ” で始まる 16 進数の数値を指定してください。

スタック・サイズ指定ファイルで，不正な加算サイズの指定が見つかりました。加算サイズは，“ ADD= ” の後に，10 進数または，“ 0x ” で始まる 16 進数の数値を指定してください。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9414: スタックサイズ指定ファイル（*path*:*line*）で，不正な加算情報がありました（*value1*,*value2*）。加算サイズは一関数に一つしか指定できません。

スタック・サイズ指定ファイルで，不正な加算サイズの指定が見つかりました。加算サイズは一関数に 1 つしか指定できません。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9415: スタックサイズ指定ファイル（*path*:*line*）で，不正な加算情報がありました（*value*）。再帰回数は “ RECTIME= ” の後に，正の 10 進数または，“ 0x ” で始まる 16 進数の数値を指定してください。

スタック・サイズ指定ファイルで，不正な加算サイズの指定が見つかりました。再帰回数は，“ RECTIME= ” の後に，正の 10 進数または，“ 0x ” で始まる 16 進数の数値を指定してください。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9416: スタックサイズ指定ファイル（*path*:*line*）で，不正な加算情報がありました（*value1*,*value2*）。再帰回数は一関数に一つしか指定できません。

スタック・サイズ指定ファイルで，不正な加算サイズの指定が見つかりました。再帰回数は一関数に 1 つしか指定できません。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9417: スタックサイズ指定ファイル（*path*:*line*）で，不正な加算情報がありました（*value*）。再帰回数の指定 “ RECTIME= ” は再帰関数にしか使うことができません。

スタック・サイズ指定ファイルで，不正な加算情報の指定が見つかりました。再帰回数の指定“ RECTIME= ” は再帰関数にしか使うことができません。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9418: スタックサイズ指定ファイル（*path*:*line*）で，不正な加算情報がありました（*value*）。呼び出し関数は“ CALL= ”の後に関数名を指定してください。

スタック・サイズ指定ファイルで，不正な加算情報の指定が見つかりました。呼び出し関数は“ CALL= ”の後に関数名を指定してください。プロジェクトで使用されていないものを登録することはできません。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9419: スタックサイズ指定ファイル（*path*:*line*）で，不正な加算情報がありました（*value*）。呼び出し関数は“ CALL= ”の後に一つの関数しか指定できません。

スタック・サイズ指定ファイルで，不正な加算情報の指定が見つかりました。呼び出し関数は“ CALL= ”の後に 1 つの関数しか指定できません。複数を指定する場合，“ CALL= ”を複数指定してください。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9423: スタックサイズ指定ファイル（*path*:*line*）で，加算情報の指定限界数を越えています。1,024 個以下にして下さい。

スタック・サイズ指定ファイルで，加算情報の指定限界を越えた数の指定があります。1,024 個以下にしてください。呼び出す関数すべてを登録してもスタック・サイズに加算されるのは，スタック・サイズが最大となる関数だけです。呼び出し関数を減らすようにしてください。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9424: スタックサイズ指定ファイル（*path*）に，同一関数に対する指定が（*line1*）と（*line2*）で見つかりました。片方を削除して下さい。

スタック・サイズ指定ファイルで，同一関数に対する指定が *line1* と *line2* で見つかりました。片方を削除してください。詳しくは，「14.6.2 スタック・サイズ指定ファイル」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9425: ファイル（*path*:*line*）は一行の文字数限界（*num*）を越えています。一行を短くしてください。

ファイル（*path*:*line*）は一行の文字数限界（*num*）を越えています。一行を短くしてください。各ファイルの限界値は，「14.5.1 stk850 の限界値」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9426: ファイル（*path*:*line*）は一行の登録限界数（*num*）を越えています。登録情報の数を減らしてください。

ファイル（*path*:*line*）は一行の登録限界数（*num*）を越えています。登録情報の数を減らしてください。各ファイルの限界値は，「14.5.1 stk850 の限界値」を参照してください。[読み込みを中止しますか？] ダイアログが出力されます。

W9427: 標準ライブラリ用のスタックサイズ変更ファイル（*path*）が見つかりませんでした。stk850 が持っているサイズ情報（libsize32.txt に同じ）を使います。

標準ライブラリ用のスタック・サイズ変更ファイル（*path*）が見つかりませんでした。stk850 が持っているサイズ情報（libsize32.txt に同じ）を使います。インストールが正しく行われていない可能性があります。

W9430: 中間アセンブラ・ファイルが見つかりませんでした。PM+ で C ソースファイルを登録し，[ツール]-[コンパイラオプションの設定] の「一般」タブで，「アセンブラソース」を指定した上で，リビルドして下さい。

中間アセンブリ言語ファイルが見つかりません。PM+ で C 言語ソース・ファイルを登録し，[コンパイラオプションの設定] ダイアログの [一般] タブで“ アセンブラソース [-Fs] ”を指定し，リビルドしてください。

W9450: 出力結果が 32,767 行を越えています。関数の選択を変更するか，最大経路のみの出力にしてください。

出力結果が 32,767 行を越えています。関数の選択を変更するか，最大経路のみの出力にしてください。

W9451: 出力結果の一行が 5,119 文字を越えています。関数の選択を変更するか，テキスト形式で出力して下さい。

出力結果の一行が 5,119 文字を越えています。関数の選択を変更するか，テキスト形式で出力してください。テキスト形式の場合，改行して表示します。

W9460: 一関数に対する加算情報数の限界（1,024）を越えています。加算情報（呼び出し関数）を減らして下さい。

関数に対する加算情報数の限界（1,024）を越えています。加算情報（呼び出し関数）を減らしてください。呼び出す関数をすべて登録してもスタック・サイズに加算されるのは，スタック・サイズが最大となる関数だけです。

W9461: 一関数に対する加算情報の文字数限界（5,119）を越えました。加算情報（呼び出し関数）を減らして下さい。

一関数に対する加算情報の文字数限界（5,119）を越えました。加算情報（呼び出し関数）を減らしてください。5,119 文字には，要素名とセパレータを含みます。

W9462: 加算サイズに不正な文字（*character*）がありました。10 進数で整数，または，“ 0x ” で始まる 16 進数で指定して下さい。

加算サイズに不正な文字がありました。10 進数で整数，または，“ 0x ”で始まる 16 進数で指定してください。

W9463: 再帰回数に不正な文字（character）がありました。10 進数で正の整数，または "0x" で始まる 16 進数で指定して下さい。

再帰回数に不正な文字がありました。10 進数で正の整数，または "0x" で始まる 16 進数で指定して下さい。

W9464: 加算サイズが大きすぎます。2,147,483,647 以下に修正してください。

加算サイズに 2,147,483,647 を越える値が指定されました。値を修正してください。

W9465: 関数（*function*）のスタック使用量が 2,147,483,647 を越えました。再帰回数の値を小さくしてください。

関数のスタック使用量が，2,147,483,647 を越えました。再帰回数の値を小さくしてください。

W9466: 関数（*function*）のスタック使用量が，2,147,483,647 を越えました。最大経路中の関数からスタックサイズを減らして下さい。

関数のスタック使用量が，2,147,483,647 を越えました。最大経路中の関数からスタック・サイズを減らしてください。

I9500: stk850 を終了します。

[OK] ボタンで，stk850 を終了します。

I9501: ファイルを上書きしますか？

[はい] ボタンで，既存のファイルに対して上書き保存をします。[いいえ] ボタンで，上書きをキャンセルします。

I9502: 関数（*function*）を初期値に戻しますか？

[はい] ボタンで，関数 *function* に対する設定を初期値に戻します。[いいえ] ボタンで，キャンセルします。

I9503: 全ての関数を初期値に戻しますか？

[はい] ボタンで，すべての関数に対する設定を初期値に戻します。[いいえ] ボタンで，キャンセルします。

付録 C　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@necel.com

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクスの販売特約店へお申し付けください。