お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

アプリケーション・ノート

CC78K4 から CA850 への移植ガイド

対象デバイス

V850 ファミリ™

対象ツール

CA850 Ver.2.40
CC78K4 Ver.2.20
RA78K4 Ver.1.30

資料番号 U15653JJ1V0AN00（第1版）
発行年月 July 2001 NS CP(K)

〔メ　モ〕

目次要約

V800 シリーズ，V850 ファミリ，V851, V852, V853, V854, V850/SA1, V850/SB1, V850/SB2, V850/SV1, V850/SF1, V850E/MS1, V850E/MS2, V850E/MA1, V850E/MA2, V850E/IA1, V850E/IA2 は，日本電気株式会社の商標です。

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。

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

対象者　このアプリケーション・ノートは78K/IVシリーズのマイクロコンピュータおよびV800シリーズ™のマイクロコンピュータについて理解されていて，78K/IVシリーズのCC78K4 Cコンパイラ（以降CC78K4），RA78K4アセンブラ・パッケージ（以降RA78K4）について理解されているユーザを対象としています。

目　的　このアプリケーション・ノートは，CC78K4用に記述されたプログラムを，V850ファミリのCA850 Cコンパイラ・パッケージ（以降CA850）用に置き換える際に，注意すべき点および基本的な記述方法を理解していただくことを目的としています。

構　成　このアプリケーション・ノートの構成は次のとおりです。

第1章　概　　説

CC78K4, RA78K4とCA850の全体的な比較について説明します。

第2章　C言語編

C言語でのコンパイラ依存の記述をCC78K4からCA850に置き換える際の説明をします。

第3章　アセンブリ言語編

アセンブラ制御命令、疑似命令をRA78K4からCA850に置き換える際の説明をします。

第4章　リンク・ディレクティブ編

リンク・ディレクティブをCC78K4からCA850に置き換える際の説明をします。

第5章　翻訳限界

CC78K4, RA78K4とCA850の最大性能について説明します。

なお，このアプリケーション・ノートではインストラクションについての説明はしていません。インストラクションについては各マイクロコンピュータのマニュアルを参照してください。

読み方　“第1章　概　　説”からお読みください。C言語でプログラムを記述しない方は“第2章　C言語編”を読み飛ばされても結構です。また，アセンブラでプログラムを記述しない方は“第3章　アセンブリ言語編”を読み飛ばされても結構です。リンク・ディレクティブ・ファイルを変更する場合には“第4章　リンク・ディレクティブ編”をお読みください。

凡　例

［　］	：［　］の中は省略可能です。
“　”	：参照先を示します。
【CC78K4】	：CC78K4での説明，記述形式の説明です。
【RA78K4】	：RA78K4での説明，記述形式の説明です。
【CA850】	：CA850での説明，記述形式の説明です。
＜記述例＞	：記述例です。

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。

78K/**IV**シリーズに関する資料（ユーザーズ・マニュアル）

資　料　名		資料番号	
		和　　文	英　　文
IE-78K4-NS		U13356J	U13356E
RA78K4　**アセンブラ・パッケージ**	**操作編**	U11334J	U11334E
	言語編	U11162J	U11162E
	構造化アセンブラ・プリプロセッサ	U11743J	U11743E
CC78K4　**Cコンパイラ**	**操作編**	U11572J	U11572E
	言語編	U11571J	U11571E
SM78K4　**システム・シミュレータ**　Ver.1.40**以上** Windows™**ベース**	**レファレンス編**	U10093J	U10093E
SM78K**シリーズ　システム・シミュレータ**　Ver.1.40**以上**	**外部部品ユーザ・オープン・インタフェース仕様編**	U10092J	U10092E
ID78K**シリーズ　統合ディバッガ**　Ver.2.30**以上** Windows**ベース**	**操作編**	U15185J	U15185E
ID78K4　**統合ディバッガ**　Windows**ベース**	**レファレンス編**	U10440J	U10440E
RX78K4　**リアルタイム**OS	**基礎編**	U10603J	U10603E
	インストール編	U10604J	U10604E
	ディバッガ編	U10364J	－

V850ファミリに関する資料（ユーザーズ・マニュアル）（ハードウエア・ツール）

資　料　名	資料番号	
	和文	英文
IE-703002-MC（V851™, V852™, V853™, V854™, V850/SA1™, V850/SB1™, V850/SB2™, V850/SV1™, V850/SF1™用インサーキット・エミュレータ）	U11595J	U11595E
IE-703003-MC-EM1（V853用インサーキット・エミュレータ・オプション・ボード）	U11596J	U11596E
IE-703008-MC-EM1（V854用インサーキット・エミュレータ・オプション・ボード）	U12420J	U12420E
IE-703017-MC-EM1（V850/SA1用インサーキット・エミュレータ・オプション・ボード）	U12898J	U12898E
IE-703037-MC-EM1（V850/SB1, V850/SB2用インサーキット・エミュレータ・オプション・ボード）	U14151J	U14151E
IE-703040-MC-EM1（V850/SV1用インサーキット・エミュレータ・オプション・ボード）	U14337J	U14337E
IE-703079-MC-EM1（V850/SF1用インサーキット・エミュレータ・オプション・ボード）	U15447J	作成予定
IE-703102-MC（V850E/MS1™, V850E/MS2™用インサーキット・エミュレータ）	U13875J	U13875E
IE-703102-MC-EM1（V850E/MS1, V850E/MS2用インサーキット・エミュレータ・オプション・ボード）， IE-703102-MC-EM1-A（V850E/MS1用インサーキット・エミュレータ・オプション・ボード）	U13876J	U13876E
IE-V850E-MC（V850E/IA1™, V850E/IA2™用インサーキット・エミュレータ）， IE-V850E-MC-A（V850E1（NB85Eコア），V850E/MA1™, V850E/MA2™用インサーキット・エミュレータ）	U14487J	U14487E
IE-V850E-MC-EM1-A（V850E1（NB85Eコア）用インサーキット・エミュレータ・オプション・ボード）	作成予定	作成予定
IE-V850E-MC-EM1-B, IE-V850E-MC-MM2（V850E1（NB85Eコア）用インサーキット・エミュレータ・オプション・ボード）	U14482J	U14482E
IE-703107-MC-EM1（V850E/MA1, V850E/MA2用インサーキット・エミュレータ・オプション・ボード）	U14481J	U14481E
IE-703116-MC-EM1（V850E/IA1用インサーキット・エミュレータ・オプション・ボード）	U14700J	作成予定
IE-703114-MC-EM1（V850E/IA2用インサーキット・エミュレータ・オプション・ボード）	作成予定	作成予定

V850ファミリに関する資料（ユーザーズ・マニュアル）（ソフトウエア・ツール）

資　料　名		資料番号	
		和文	英文
CA850　Cコンパイラ・パッケージ　Ver.2.40以上	操作編	U15024J	U15024E
	C言語編	U15025J	U15025E
	プロジェクト・マネージャ編	U15026J	U15026E
	アセンブリ言語編	U15027J	U15027E
ID850　統合ディバッガ　Ver.2.40　Windowsベース	操作編	U15181J	作成予定
ID850NW　統合ディバッガ　Ver.1.10以上 Windowsベース	操作編	U14891J	U14891E
SM850　システム・シミュレータ　Ver.2.40 Windowsベース	操作編	U15182J	作成予定
SM850　システム・シミュレータ　Ver.2.00以上	外部部品ユーザ・オープン・インタフェース仕様編	U14873J	U14873E
RX850　リアルタイムOS　Ver.3.13以上	基礎編	U13430J	U13430E
	インストレーション編	U13410J	U13410E
	テクニカル編	U13431J	U13431E
RX850 Pro　リアルタイムOS　Ver.3.13	基礎編	U13773J	U13773E
	インストレーション編	U13774J	U13774E
	テクニカル編	U13772J	U13772E
RX-NET　TCP/IPライブラリ		U15083J	－
RD850　タスク・ディバッガ　Ver.3.01		U13737J	U13737E
RD850 Pro　タスク・ディバッガ　Ver.3.01		U13916J	U13916E
AZ850　システム・パフォーマンス・アナライザ　Ver.3.0		U14410J	U14410E
PG-FP3　フラッシュ・メモリ・プログラマ		U13502J	U13502E
CA850　Cコンパイラ・パッケージ　Ver.2.40 （アプリケーション・ノート）	コーディング・テクニック	U15184J	U15184E
V800シリーズ開発ツール（32ビット対応）　Ver.2.40 Windowsベース（アプリケーション・ノート）	チュートリアル・ガイド	U15196J	U15196E
CC78K4からCA850への移植ガイド（アプリケーション・ノート）		このマニュアル	作成予定

目　次

図の目次

表の目次

第1章　概　　説

この章では，CC78K4およびRA78K4とCA850の概要について説明します。

1.1　製品形態

78K/IVシリーズの場合には，Cコンパイラがcc78K4，アセンブラとリンカがRA78K4として別製品となっていますが，V850ファミリのCA850では1つの製品の中にCコンパイラ，アセンブラ，リンカがパッケージされています。

表1－1　製品名

	78K/IVシリーズ	V850ファミリ
Cコンパイラ・パッケージ	USxxxxCC78K4	USxxxxCA703000
アセンブラ・パッケージ	USxxxxRA78K4	

ただし，デバイス・ファイルに関しては，78K/IVシリーズ，V850ファミリともに別途購入していただく必要があります。

1.2 パッケージ・ソフト

CC78K4，RA78K4とCA850の製品には，次のようなプログラムが含まれています。

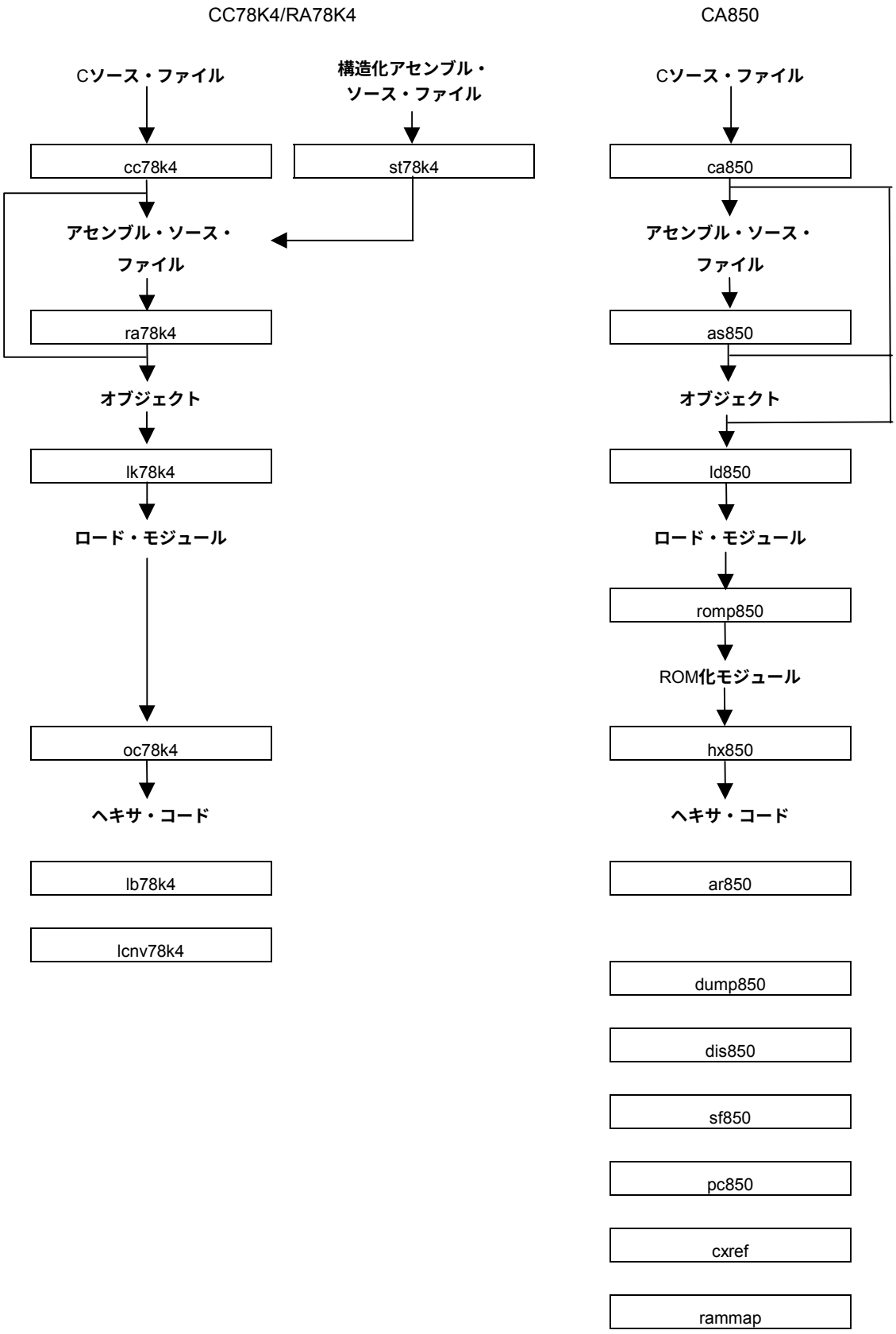
表1－2　パッケージ・ソフト

名称[注]	CC78K4，RA78K4	CA850
Cコンパイラ	cc78k4	ca850
アセンブラ	ra78k4	as850
リンカ	lk78k4	ld850
ROM化プロセッサ	－	romp850
オブジェクト・コンバータ ヘキサ・コンバータ	oc78k4	hx850
ライブラリアン アーカイバ	lb78k4	ar850
構造化アセンブラ	st78k4	－
リスト・コンバータ	lcnv78k4	－
ダンプ・コマンド	－	dump850
ディス・アセンブラ	－	dis850
セクション・ファイル・ジェネレータ	－	sf850
パフォーマンス・チェッカ	－	pc850
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

注　名称が2段になっているものは，上段がRA78K4での名称，下段がCA850での名称になります。

各プログラムの開発手順は次のようになります。

図1－1　開発手順



第2章　C言語編

この章では，Cソース・プログラムをCC78K4用プログラムからCA850用プログラムに置き換える場合の注意する点，および基本的な記述方法について説明します。

CC78K4とCA850はANSI-C[注]規格に準拠しています。

注　ANSIとはAmerican National Standards Instituteの略称です。

2. 1　コンパイラ定義のマクロ

CC78K4，CA850では，次のマクロがあらかじめ定義されています。

表2－1　コンパイラ定義マクロ

CC78K4，RA78K4	CA850	内容
__LINE__		カレント・ソース行の行番号
__FILE__		ソース・ファイル名
__DATE__		ソース・ファイルのコンパイル日付（"mm dd yyyy"）
__TIME__		ソース・ファイルのコンパイル時刻（"hh:mm:ss"）
__STDC__		10進定数の1 （CC78K4：-ZAオプション指定時に定義 CA850 ：-ansiオプション指定時に定義）
__K4__	__v850 __v850__	10進定数の1 （デバイスのシリーズ名を示すマクロ）
CPUマクロ		10進定数の1（ターゲットのCPUを示すマクロ） μPD784038，μPD703003の場合の例を示します
__4038_	__3003__	

このコンパイラ定義マクロを利用することにより，CC78K4の記述とCA850の記述をCソース・プログラム上で混在させることが可能になります。

＜記述例＞

```
#ifdef __K4__
        #pragma sfr                 /* CC78K4用の記述 */
#endif
#ifdef __v850__
        # pragma ioreg              /* CA850用の記述 */
#endif
```

2. 2　#pragma指令

#pragma指令は，コンパイラに対し，コンパイラ定義の方法で動作することを指示する指令です。CC78K4とCA850の#pragma指令には次のようなものがあります。

表2－2　#pragma指令

項番	機能	CC78K4	CA850
1	特殊機能レジスタ名の利用	#pragma sfr	#pragma ioreg
2	アセンブラ命令の記述	#pragma asm	#pragma asm #pragma endasm
3	割り込み関数	#pragma vect #pragma interrupt	#pragma interrupt
4	割り込み禁止関数指定	関数の先頭でDI();を記述すると前処理よりも前に割り込み禁止になる[注]	#pragma block_interrupt
5	割り込み禁止制御機能（DI/EI）	#pragma di #pragma ei	#pragma指令は必要なし
6	CPU制御命令	#pragma halt #pragma stop #pragma nop #pragma brk	#pragma指令はなし アセンブラ命令記述
7	絶対番地アクセス	#pragma access	なし
8	セクション名の変更	#pragma section	#pragma section #pragma text
9	モジュール名の変更	#pragma name	なし
10	インライン展開指定	なし	#pragma inline
11	ローテート関数の使用	#pragma rot	なし
12	乗算関数の使用	#pragma mul	なし
13	除算関数の使用	#pragma div	なし
14	データ挿入関数の使用	#pragma opc	#pragma指令はなし アセンブラ命令記述
15	リアルタイムOS対応 割り込みハンドラ指定	#pragma rtos_interrupt	#pragma指令はなし アセンブラ記述
16	リアルタイムOS関数指定	#pragma rtos_task	#pragma rtos_task
17	デバイス種別指定	#pragma pc	#pragma cpu
18	構造体パッキング	なし	#pragma pack

注　DI();を記述する場合には，#pragma di指令が必要です。

2. 2. 1　特殊機能レジスタ名（周辺機能レジスタ名）の利用

CC78K4からCA850に置き換える際には，“ sfr ”を“ ioreg ”に変更してください。

【CC78K4】

#pragma指令により，Cソース中に特殊機能レジスタ名を使用することを宣言します。

```
#pragma sfr
```

【CA850】

#pragma指令により，Cソース中に周辺機能レジスタ名を使用することを宣言します。

```
#pragma ioreg
```

2. 2. 2　アセンブラ命令の記述

Cソース・プログラム中にアセンブラ命令を記述する方法はCC78K4，CA850ともに，2通りの方法があります。

【CC78K4】

（1）1行挿入の場合

#pragma指令により，__asmの使用を宣言します。

```
#pragma asm
```

Cソース中に次の形式で記述します。

```
__asm(文字列リテラル);
```

＜記述例＞

```
#pragma asm
void main(void)
{
      __asm("¥tMOV¥tA,B");
}
```

（2）複数行挿入の場合

#asmでアセンブラ命令の開始を示し，#endasmでアセンブラ命令の終了を示します。

アセンブラ命令は#asm，#endasmの間に記述します。

```
#asm
  アセンブラ命令
#endasm
```

＜記述例＞

```
#asm
        MOV       A,B
#endasm
```

【CA850】

（1）1行挿入の場合

Cソース中に次の形式で記述します。

__asm(**文字列定数**);

または

_asm(**文字列定数**);

ただし，-ansi**オプションを指定する場合，**__asm**形式を使用してください。**

＜記述例＞

```
__asm("¥tmov¥tr10,r11");
```

（2）複数行挿入の場合

#pragma asm**でアセンブラ命令の開始を示し，**#pragma endasm**でアセンブラ命令の終了を示します。**

アセンブラ・ソースは#pragma asm**，**#pragma endasm**指令の間に記述します。**

```
#pragma asm
  アセンブラ命令
#pragma endasm
```

＜記述例＞

```
#pragma asm
        mov     r10,r11
#pragma endasm
```

2. 2. 3　割り込み関数

#pragma指令を使用して割り込み関数を定義する際には，割り込みベクタ（ハンドラ）にCC78K4の場合には割り込み関数のアドレス，CA850の場合には割り込み関数への分岐命令が出力されます。

CC78K4からCA850に置き換える際には，割り込み要求名の違い（マイクロコンピュータに依存します）に注意してください。

【CC78K4】

#pragma指定により割り込み要求名，関数名，スタック切り替え，レジスタおよび使用するsaddr2領域の退避／復帰を指定します。

#pragma vect割り込み要求名 関数名 ［,スタック切り替え指定］
スタック使用指定 ［｛無変更指定｝］ レジスタ・バンク指定

または

#pragma interrupt割り込み要求名 関数名 ［,スタック切り替え指定］
スタック使用指定 ［｛無変更指定｝］ レジスタ・バンク指定

割り込み関数修飾子は2種類存在します。

ノンマスカブル／マスカブル割り込み関数の場合

__interrupt 関数定義 または 関数宣言

ソフトウエア割り込み関数の場合

__interrupt_brk 関数定義 または 関数宣言

＜記述例＞

```
#pragma interrupt INTP0 int1
#pragma vect INTP1 int2 sp=buff+10 RB1
#pragma interrupt BRK_I int_b RB2

__interrupt
void int1(void)
{
        …
}
__interrupt
void int2(void)
{
        …
}
__interrupt_brk
void int_b(void)
{
        …
}
```

【CA850】

#pragma**指定により割り込み要求名，関数名，配置方法を指定します。**

#pragma interrupt**割り込み要求名 関数名 ［配置方法］**

割り込み関数修飾子は次のようになります。

__interrupt **関数定義 または 関数宣言**

ただし，多重割り込みの場合の修飾子は次のようになります。

__multi_interrupt **関数定義 または 関数宣言**

＜記述例＞

```
#pragma interrupt INTP110 int1
#pragma interrupt INTP111 int2 direct
#pragma interrupt INTP112 int3

__interrupt
void int1(void)
{
        …
}
__interrupt
void int2(void)
{
        …
}
__multi_interrupt
void int3(void)
{
        …
}
```

2. 2. 4　割り込み禁止関数指定

関数全体を割り込み禁止にします。

CC78K4からCA850に移植する際には，割り込み機能関数DI()，EI()**を使用するのではなく，**#pragma**指令で割り込み禁止とする関数名を指定してください。**

【CC78K4】

関数の先頭でDI()**を記述することにより，前処理よりも前に割り込み禁止になります。**

＜記述例＞

```
#pragma DI
#pragma EI

void func1(void)
{
        DI();
        …
        EI();
}
```

【CA850】

#pragma**指令により，関数名を指定します。**

#pragma block_interrupt **関数名**

＜記述例＞

```
#pragma block_interrupt func1
void func1(void)
{
        …
}
```

2. 2. 5　割り込み禁止制御機能

割り込み機能関数は，CC78K4，CA850ともにあります。ただし，この関数を使用する際には，CC78K4では#pragma指令が必要ですが，CA850では必要ありません。

【CC78K4】

#pragma指令により，DI()，EI()を割り込み機能関数として使用することを指定します。

```
#pragma di
#pragma ei
```

関数呼び出しと同様の形式でＣソース中に記述します。

```
DI();
EI();
```

＜記述例＞

```
#pragma DI
#pragma EI

void func2(void)
{
        …
        DI();
        …
        EI();
        …
}
```

【CA850】

割り込み禁止制御を行う関数を提供しています。#pragma指令は必要ありません。

```
__DI();
__EI();
```

＜記述例＞

```
void func2(void)
{
        …
        __DI();
        …
        __EI();
        …
}
```

2. 2. 6　CPU制御命令

CC78K4では，Cソース・プログラム上でCPUを制御するCPU制御命令（HALT，STOP，BRK，NOP）があります。

CA850にはCPU制御命令はありませんので，アセンブラ命令で記述してください。

【CC78K4】

#pragma指令により，HALT()，STOP()，BRK()，NOP()を割り込み機能関数として使用することを指定します。

```
#pragma halt
#pragma stop
#pragma brk
#pragma nop
```

関数呼び出しと同様の形式でCソース中に記述します。

```
HALT();
STOP();
BRK();
NOP();
```

＜記述例＞

```
#pragma HALT
#pragma STOP
#pragma BRK
#pragma NOP

HALT();
STOP();
BRK();
NOP();
```

【CA850】

CPU制御命令はありません。アセンブラ命令で記述してください。

halt，trap，nop

＜記述例＞

```
__asm("¥thalt");
__asm("¥ttrap¥t0x00");
__asm("¥tnop");
```

2.2.7　絶対番地アクセス

CC78K4には，絶対番地アクセス用の関数があります。

CA850では絶対番地アクセス用の関数がありませんので，ポインタを使用して絶対番地にアクセスしてください。

【CC78K4】

#pragma指令により，絶対番地アクセス用の関数を使用することを指定します。

#pragma access

関数呼び出しと同様の形式でCソース中に記述します。

絶対番地アクセス用の関数名は次の4つです。

peekb　：引き数のアドレスの内容を1バイト返します。

peekw　：引き数のアドレスの内容を2バイト返します。

pokeb　：引き数のアドレスに1バイト書き込みます。

pokew　：引き数のアドレスに2バイト書き込みます。

＜記述例＞

```
#pragma access

int     data1,data2;
char    cdata1,cdata2;

cdata1 = peekb ( 0x1234 );
data1 = peekw ( 0x1234 );
pokeb ( 0xfe20 , 5 );
pokew ( 0xfe20 , 0xffff );
```

【CA850】

絶対番地アクセス用の関数はありません。次のようにポインタを使用してください。

または，記述例2のサンプルを参考にしてください。

＜記述例1＞

```
int             data1,data2;
char    cdata1,cdata2;
char    *p;
int     *q;

cdata1 = *((char *)0x1234);
data1 = *((int *)0x1234);

p = (char *)0xffe000;
*p = 5;
q = (int *)0xffe000;
*q = 0xffff;
```

＜記述例2＞

```
unsigned char peekb(unsigned int);
unsigned short peekh(unsigned int);
unsigned int peekw(unsigned int);

void pokeb(unsigned int,unsigned char);
void pokeh(unsigned int,unsigned short);
void pokew(unsigned int,unsigned int);

unsigned char peekb(unsigned int adr)
{
        return *((unsigned char *)adr);
}

unsigned short peekh(unsigned int adr)
{
        return *((unsigned short *)adr);
}

unsigned int peekw(unsigned int adr)
{
        return *((unsigned int *)adr);
}

void pokeb(unsigned int adr,unsigned char val)
{
unsigned char *p;

        p = (unsigned char *)adr;
        *p = val;
}

void pokeh(unsigned int adr,unsigned short val)
{
unsigned short *p;

        p = (unsigned short *)adr;
        *p = val;
}

void pokew(unsigned int adr,unsigned int val)
{
unsigned int *p;

        p = (unsigned int *)adr;
        *p = val;
}
```

2. 2. 8　セクション名の変更

#pragma指令のsectionは同じですが，その後に指定する文字列は違いますので，注意してください。

【CC78K4】

#pragma指令により，変更するセクション名と変更後のセクション名およびセクションの開始アドレスを指定します。

#pragma section コンパイラ出力セクション名 変更セクション名

［AT 開始アドレス］

＜記述例＞

```
#pragma section @@CODE CODESEG
        /* セグメント名@@CODEをCODESEGに変更 */
#pragma section @@DATA DATASEG AT 0FFE10H
        /* セグメント名@@DATAをDATASEGに変更し0FFE10H 番地に配置 */
```

【CA850】

#pragma指令により，データやモジュールにセクション名を指定します。

セクション名を省略した場合は，ディフォールトのセクション名になります。

データの場合

#pragma section セクション種別 ［"セクション名"］ begin

変数の宣言／定数

#pragma section セクション種別 ［"セクション名"］ end

モジュールの場合

#pragma text ［"セクション名"］［関数名］

割り当てを指定できるデータ・セクション種別は次のとおりです。

.tidata（.tidataセクション，.tibssセクション）

.data（.dataセクション，.bssセクション）

.sdata（.sdataセクション，.sbssセクション）

.sedata（.sedataセクション，.sebssセクション）

.sidataセクション（.sidataセクション，.sibssセクション）

.sconstセクション

.constセクション

ただし，セクション名を指定できるセクション種別は次のとおりです。

.data（.dataセクション，.bssセクション）

.sdata（.sdataセクション，.sbssセクション）

.constセクション

.sconstセクション

＜記述例＞

```
#pragma section sconst begin
const int const_data=10;
#pragma section sconst end
        /* const_data変数を.sconstセクションに配置 */
#pragma section data "d1sec" begin
int data_data;
#pragma section data "d1sec" end
        /* data_data変数をdata属性のd1sec.bssセクションに配置 */

#pragma text "f1sec" func1
        /* func1関数をf1sec.textセクションに配置 */
```

2. 2. 9　モジュール名の変更

CA850では，モジュール名を変更することはできません。

【CC78K4】

#pragma指令によりモジュール名を指定します。

#pragma name モジュール名

＜記述例＞

```
#pragma name new_name
```

【CA850】

ありません。

2. 2. 10　インライン展開指定

CC78K4ではユーザの定義した関数をインライン展開することはできませんが，CA850では#pragma指令によりユーザ関数のインライン展開の指定が可能です。

【CC78K4】

ありません。

【CA850】

#pragma指令により，関数ごとのインライン展開を指定します。

#pragma inline 関数名　[,関数名　…]

＜記述例＞

```
#pragma inline func
```

2. 2. 11　ローテート関数の使用

V850ファミリではローテートのインストラクションがありません。そのため，CA850にローテート関数はありません。ローテートを使用したい場合には，記述例を参考にし，アセンブラのプログラムを作成してください。

【CC78K4】

#pragma指令により，ローテートするコードを関数呼び出しではなく，直接インライン展開して出力することを指定します。

#pragma rot

関数呼び出しと同様の形式でＣソース中に記述します。

ローテートする関数名は次の4つです。

rorb

rolb

rorw

rolw

＜記述例＞

```
#pragma rot

ucdata3 = rorb(ucdata1,ucdata2);
ucdata3 = rolb(ucdata1,ucdata2);
uidata3 = rorw(uidata1,ucdata2);
uidata3 = rolw(uidata1,ucdata2);
```

【CA850】

ローテート関数はありません。変数をロ－テートさせるには，次のサンプルを参考にして，プログラムを作成してください。ただし，インライン展開はされません。

＜記述例＞

Ｃソースでのプロトタイプ宣言

```
unsigned char rorb(unsigned char,unsigned char);
unsigned char rolb(unsigned char,unsigned char);
unsigned short rorh(unsigned short,unsigned char);
unsigned short rolh(unsigned short,unsigned char);
unsigned int rorw(unsigned int,unsigned char);
unsigned int rolw(unsigned int,unsigned char);
```

アセンブラでの記述（1/2）

```
        .globl   _rorb
        .globl   _rolb
        .globl   _rorh
        .globl   _rolh
        .globl   _rorw
        .globl   _rolw

        .file    "rot.s"
        .text
-- 1byte variable rotation --
        .align   4
_rorb:
        shr      1,r6
        bnc      rorb1
        or       0x00000080,r6
rorb1:
        add      -1,r7
        mov      r6,r10
        bnz      _rorb
        jmp[lp]

        .align   4
_rolb:
        add      -4,sp
rolb1:
        shl      1,r6
        st.h     r6,[sp]
        tst1     0,1[sp]
        bz       rolb2
        or       0x00000001,r6
rolb2:
        add      -1,r7
        mov      r6,r10
bnz     rolb1
add     4,sp
jmp[lp]


-- 2byte variable rotation --
        .align   4
_rorh:
                 shr      1,r6
        bnc      rorh1
                 or       0x00008000,r6
rorh1:
                 add      -1,r7
        mov      r6,r10
                 bnz      _rorh
        jmp[lp]

                 .align   4
_rolh:
                 add      -4,sp
```

アセンブラでの記述（2/2）

```
 rolh1:
                  shl      1,r6
         st.w     r6,[sp]
         tst1     0,2[sp]
                  bz       rolh2
         or       0x00000001,r6
rolh2:
         add      -1,r7
         mov      r6,r10
         bnz      rolh1
                  add      4,sp
         jmp[lp]


-- 4byte variable rotation --
         .align   4
_rorw:
                  shr      1,r6
         bnc      rorw1
         or       0x80000000,r6
rorw1:
         add      -1,r7
         mov      r6,r10
         bnz      _rorw
         jmp[lp]

         .align   4
_rolw:
         shl      1,r6
         bnc      rolw1
         or       0x00000001,r6
rolw1:
         add      -1,r7
         mov      r6,r10
         bnz      _rolw
         jmp[lp]
```

2. 2. 12　乗算関数の使用

CA850では乗算関数はありませんので，式の値で記述してください。

【CC78K4】

#pragma指令により，乗算するコードを関数呼び出しではなく，直接インライン展開して出力することを指定します。

#pragma mul

関数呼び出しと同様の形式でＣソース中に記述します。

乗算する関数名は次の3つです。

mulu

muluw

mulw

＜記述例＞

```
#pragma mul
uidata3 = mulu(ucdata1,ucdata2);
uldata3 = muluw(uidata1,uidata2);
ldata3 = mulw(idata1,idata2);
```

【CA850】

インライン展開する機能はありません。

関数の形ではなく，式の形（*演算子）で記述してください。

ただし，int型，long型が4バイトになりますので，muluw，mulwの引き数に入れる引き数のサイズには注意してください（int型はshort型にしてください）。

2. 2. 13　除算関数の使用

CA850では除算関数はありませんので，式の値で記述してください。

【CC78K4】

#pragma指令により，除算するコードを関数呼び出しではなく，直接インライン展開して出力することを指定します。

#pragma div

関数呼び出しと同様の形式でＣソース中に記述します。

除算する関数名は次の2つです。

divuw

moduw

＜記述例＞

```
#pragma div
uidata3 = divuw(uidata1,ucdata2);
ucdata3 = moduw(uidata1,ucdata2);
```

【CA850】

インライン展開する機能はありません。

関数の形ではなく，式の形（/演算子または%演算子）で記述してください。

2. 2. 14　データ挿入関数の使用

CA850ではデータ挿入関数はありませんので，アセンブラで記述してください。

【CC78K4】

#pragma指令により，データ挿入用の関数を使用することを指定します。

#pragma opc

関数呼び出しと同様の形式でＣソース中に記述します。

__OPC(データ値［, データ値　…］)

＜記述例＞

```
#pragma opc

__OPC(0xBF,0x12);
```

【CA850】

データ挿入用の関数はありません。アセンブラ疑似命令で記述してください。

＜記述例＞

```
__asm(".byte 0xBF,0x12");
```

2. 2. 15　リアルタイムOS対応割り込みハンドラ指定

CA850では直接起動割り込みハンドラの記述をＣ言語で記述することはできません。アセンブラで記述する必要があります。

【CC78K4】

#pragma指令により割り込み要求名，関数名，スタック切り替えを指定します。

#pragma rtos_interrupt 割り込み要求名 関数名 ［,スタック切り替え指定］

割り込み関数修飾子は次のようになります。

ノンマスカブル／マスカブル割り込み関数の場合

__rtos_interrupt 関数定義 または 関数宣言

＜記述例＞

```
#pragma rtos_interrupt INTP3 int4 sp=buff+10
__rtos_interrupt
void int4(void)
{
        …
        ret_int();
}
```

【CA850】

#pragma指令はありまん。アセンブラで記述する必要があります。

RX850とRX850Proには，直接起動割り込みハンドラ用のマクロが用意されていますので，直接起動割り込みハンドラは次のように記述してください。

また，RX850またはRX850Proでは復帰の処理によって記述方法が複数ありますので，詳細はリアルタイムOSのユーザーズ・マニュアルを参照してください。

＜記述例＞

RX850でretiで復帰する場合

```
#include"stdrx.inc"
    .section    " INTP123"
    jr  _inthdr
    .text
    .align 4
    .globl      _inthdr
_inthdr:
    RTOS_IntEntry
        …
    RTOS_IntExit
```

2. 2. 16　リアルタイムOS関数指定

記述形式は，CC78K4とCA850で違いはありません。

【CC78K4】

#pragma指令により，指定された関数名をリアルタイムOS用タスクとして解釈します。

#pragma rtos_task タスク関数名

【CA850】

#pragma指令により，指定された関数名をリアルタイムOS用タスクとして解釈します。

#pragma rtos_task タスク関数名

CC78K4とCA850で違いはありません。

＜記述例＞

```
#pragma rtos_task func4
void func4(void)
{
        …
        ext_tsk();
}
```

2. 2. 17　デバイス種別指定

CC78K4からCA850に置き換える際には，”pc”を”cpu”に変更してください。

また，デバイスを指定する際には，かっこではなく空白になりますので注意してください。

【CC78K4】

#pragma指令により，デバイス種別を指定します。

#pragma pc(デバイス種別)

＜記述例＞

```
#pragma pc(4038)
```

【CA850】

#pragma指令により，デバイス種別を指定します。

#pragma cpu デバイス種別

＜記述例＞

```
#pragma cpu 3003
```

2. 2. 18　構造体パッキング

CC78K4には，構造体のパッキング機能はありません。構造体は詰めた状態で領域が確保されます。

【CC78K4】

ありません。

【CA850】

#pragma指令により，指定された値をカレントのパッキング値とします。指定した値は，次の#pragma pack指令が現れるまでその指定が有効になります。

#pragma pack（パッキング値）

＜記述例＞

```
#pragma pack(1)
struct{
     char data1;
     short data2;
     char data3;
}
```

2. 3　拡張記述

デバイス固有の機能を実現するために拡張記述があります。CC78K4とCA850の拡張記述は次のようになります。

表2－3　拡張記述

項番	機能	CC78K4	CA850
1	callt関数	callt / __callt	なし
2	レジスタ変数	register	register
3	saddr領域利用	sreg / __sreg / __sreg1	なし
4	noauto関数	noauto	なし
5	norec関数	norec	なし
6	ビット型変数	bit/boolean / __boolean / __boolean1	ビット・フィールド
7	ビット・アクセス	変数名.ビット位置	共用体とビット・フィールド
8	callf関数	callf / __callf	なし
9	2進定数	0bxxxxxxxx	0bxxxxxxxx
10	割り込みレベル	なし	__set_il
11	パスカル関数	__pascal	なし

2. 3. 1　callt関数

CA850ではV850Eのみcallt命令があります。ただし，CA850では通常の関数にしてください。

【CC78K4】

callt / __calltはcalltという領域（0x40から0x7f）に，呼び出す関数のアドレスを格納し，関数を直接呼び出すよりも短いコードで呼び出すことを可能にします。

呼び出しにはcallt命令を使用します。

記述形式は次のとおりになります。

callt　型名　関数名

__callt　型名　関数名

＜記述例＞

```
callt void func(void)
{
        …
}
```

【CA850】

V850Eにはcallt命令がありますが，コンパイラでは，関数のプロローグ／エピローグのランタイム化の場合にのみ，callt命令を使用します。

そのため，関数は通常の関数にしてください。

＜記述例＞

```
void func(void)
{
        …
}
```

2. 3. 2　レジスタ変数

CC78K4，CA850ともにレジスタ変数の記述は変わりませんが，変数のレジスタへの割り当て方が変わります。CA850では最適化を指定するとregister宣言した変数でもレジスタに割り付けられない場合もありますので，注意してください。

【CC78K4】

register宣言した変数をレジスタ（RP3，VP），saddr2領域に割り当てます。

-ZOオプション指定時には，宣言された順番に割り当てられますが，-ZOオプション未指定時には参照回数順に割り当てを行います。

記述形式は次のとおりになります。

register　型名　変数名

【CA850】

register宣言した変数をregister宣言順にレジスタ変数用のレジスタに割り当てます。ただし，最適化を指定した場合には参照回数の少ないものは割り当てられない場合があります。

記述形式は次のとおりになります。

register　型名　変数名

＜記述例＞

```
int func(void)
{
register int reg_a,reg_b;
        …
}
```

2. 3. 3　saddr領域の利用

78K/IVシリーズのsaddr2領域は，CA850の.tidata / .tibssセクションに配置することを推奨いたします。.tidataセクションは２バイト命令（sld/sst）でアクセス可能なセクションです。

【CC78K4】

sreg宣言あるいは__sreg宣言された外部変数および関数内static変数は自動的にsaddr2領域にリロケータブルに割り当てられます。

また，__sreg1宣言された変数は，自動的にsaddr1領域にリロケータブルに割り当てられます。

記述形式は次のとおりになります。

sreg　型名　変数名

__sreg　型名　変数名

__sreg1　型名　変数名

＜記述例＞

```
sreg intsreg_data0,sreg_data1,sreg_data2;
```

【CA850】

ありません。

#pragma section指令により，外部変数を内部RAMの.tidata / .tibssセクションに割り当ててください。

#pragma　section　tidata　begin

型名　変数名

#pragma　section　tidata　end

＜記述例＞

```
#pragma section tidata begin
int     sreg_data0,sreg_data1,sreg_data2;
#pragma section tidata end
```

2. 3. 4　noauto関数

CA850にnoauto関数はありませんので，通常の関数にしてください。

【CC78K4】

noauto関数はオートマティック変数に制限を設けて，前後処理（スタック・フレームの形成）のコードを出力しないようにします。

記述形式は次のとおりになります。

noauto　型名　関数名

＜記述例＞

```
noauto void func(void)
{
        …
}
```

【CA850】

noauto関数はありません。

通常の関数にしてください。

＜記述例＞

```
void func(void)
{
        …
}
```

2. 3. 5　norec関数

CA850にnorec関数はありませんので，通常の関数にしてください。

【CC78K4】

関数自身から他の関数を呼び出さない関数はnorec関数にすることができます。

norec関数では関数の前後処理（スタック・フレームの形成）のコードを出力しません。引き数は，すべてレジスタ，norec関数の引き数用saddr2領域に割り当てます。

記述形式は次のとおりになります。

norec　型名　関数名

＜記述例＞

```
norec void func(void)
{
        …
}
```

【CA850】

norec関数はありません。

通常の関数にしてください。

＜記述例＞

```
void func(void)
{
        …
}
```

2. 3. 6　ビット型変数

CA850ではビット型の変数を定義することはできませんので，ビット・フィールドを使用して記述してください。

【CC78K4】

bit，boolean，__boolean型変数は1ビットのデータとして扱われsaddr2領域に配置します。

__boolean1型変数は1ビットのデータとして扱われsaddr1領域に配置します。

bit，boolean，__ boolean，__ boolean1型変数は初期値なし（不定）の外部変数と同様に扱います。

記述形式は次のとおりになります。

bit　変数名

boolean　変数名

__boolean　変数名

__boolean1　変数名

＜記述例＞

```
boolean bit1,bit2,bit3;
void func(void)
{
        bit1 = 1;
        bit2 = bit3;
}
```

【CA850】

ありません。

ビット・フィールドを使用してください。

```
struct{
        unsigned char   f0:1;
        unsigned char   f1:1;
        unsigned char   f2:1;
        unsigned char   f3:1;
        unsigned char   f4:1;
        unsigned char   f5:1;
        unsigned char   f6:1;
        unsigned char   f7:1;
};
```

＜記述例＞

```
struct  bitf{
        unsigned int bit1:1;
        unsigned int bit2:1;
        unsigned int bit3:1;
}bitfield;

void func(void)
{
        bitfield.bit1 = 1;
        bitfield.bit2 = bitfeeld.bit3;
}
```

2. 3. 7　ビット・アクセス

CA850**ではビット・アクセスを型の変数で定義することはできませんので，ビット・フィールドを使用して記述してください。**

【CC78K4】

記述形式は次のとおりになります。

変数名.ビット位置

＜記述例＞

```
unsigned char data;
void func(void)
{
      data = 0xff;
      data.1 = 0;
}
```

【CA850】

ありません。

共用体とビット・フィールドを使用してください。

＜記述例＞

```
union{
    unsigned char cdata;
    struct{
           unsigned char bit0:1;
           unsigned char bit1:1;
           unsigned char bit2:1;
           unsigned char bit3:1;
           unsigned char bit4:1;
           unsigned char bit5:1;
           unsigned char bit6:1;
           unsigned char bit7:1;
    }bitfield;
}data;

void func(void)
{
    data.cdata = 0xff;
    data.bitfield.bit1 = 0;
}
```

2. 3. 8　callf**関数**

V850ファミリでcallf**命令に相当するインストラクションはありませんので，通常の関数にしてください。**

【CC78K4】

callf / __callf**は**callf**命令により**call**命令よりも短いコードで関数で呼ぶことを可能にします。**

記述形式は次のとおりになります。

callf　**型名　関数名**

__callf　**型名　関数名**

＜記述例＞

```
callf void func(void)
{
        …
}
```

【CA850】

callf命令はありません。

通常の関数にしてください。

＜記述例＞

```
void func(void)
{
        …
}
```

2. 3. 9　2進数定数

CC78K4とCA850で，記述形式は同じです。

【CC78K4】【CA850】

整数定数が記述可能な位置に2進定数が記述できます。

記述形式は次のとおりになります。

0b 2進定数

0B 2進定数

＜記述例＞

```
ucdata1 = 0b00010001;
ucdata2 = 0B11110000;
```

2. 3. 10　割り込みレベル指定

CA850では，割り込みレベルの指定が記述可能です。

【CC78K4】

ありません。

【CA850】

割り込みレベル（INTレベル）を制御するための関数です。

割り込みの優先順位レベルとして指定できる値は-1～8の整数値です。

“割り込み要求名”にはデバイス・ファイルに定義されているマスカブル割り込みの割り込み要求名を指定します。

割り込みの優先順位レベルとして-1を指定した場合は，マスカブル割り込みの受付を禁止し，0を指定した場合はマスカブル割り込みの受付を許可します。

記述形式は次のとおりになります。

__set_il(割り込みの優先順位レベル，"割り込み要求名")

なお，割り込みの優先順位レベルの値は，次のような意味になります。

-1	：マスカブル割り込みの受け付けを禁止
0	：マスカブル割り込みの受け付けを許可
1～8	：割り込みの優先順位レベル0～7を設定

レベルの数値に1を足したものが指定値になりますので注意してください。

＜記述例＞

割り込み要求名INTP110の優先順位レベルを1とします。

```
__set_il(2,"INTP110");
```

2. 3. 11　パスカル関数

CA850にパスカル関数はありませんので，通常の関数にしてください。

【CC78K4】

パスカル関数は，関数呼び出し時に引き数の積み込みによって使用したスタック修正を，関数呼び出し側では行われずに，呼ばれた関数側で行うコードを生成します。

関数の宣言時に__pascal属性を先頭に追加して使用します。

＜記述例＞

```
__pascal void func(void);

void func(void)
{
        …
}
```

【CA850】

パスカル関数はありません。

通常の関数にしてください。

＜記述例＞

```
void func(void);

void func(void)
{
        …
}
```

2.4　変数のサイズと整列条件

CC78K4とCA850では，同じ型でもRAMにとられる領域のサイズや変数の整列条件には違いがあります。

2.4.1　変数の型のサイズ

CC78K4とCA850の変数の型によるサイズの違いは次のようになります。

表2－4　型のサイズの差分

型	CC78K4	CA850
char	1バイト	1バイト
short	2バイト	2バイト
int[注]	2バイト	4バイト
long	4バイト	4バイト
float	4バイト	4バイト
double	4バイト	4バイト

注　int型のサイズが違いますので，注意してください。

2.4.2　整列条件

CC78K4とCA850の変数の整列条件には，次のような違いがあります。

【CC78K4】

表2－5　整列条件（CC78K4）

オプション	整列条件
ディフォールト	バイト境界整列
-RAオプション	2バイト以上の外部変数（saddr領域に割り当てられる変数をのぞく）は2バイト境界整列

【CA850】

表2－6　整列条件（CA850）

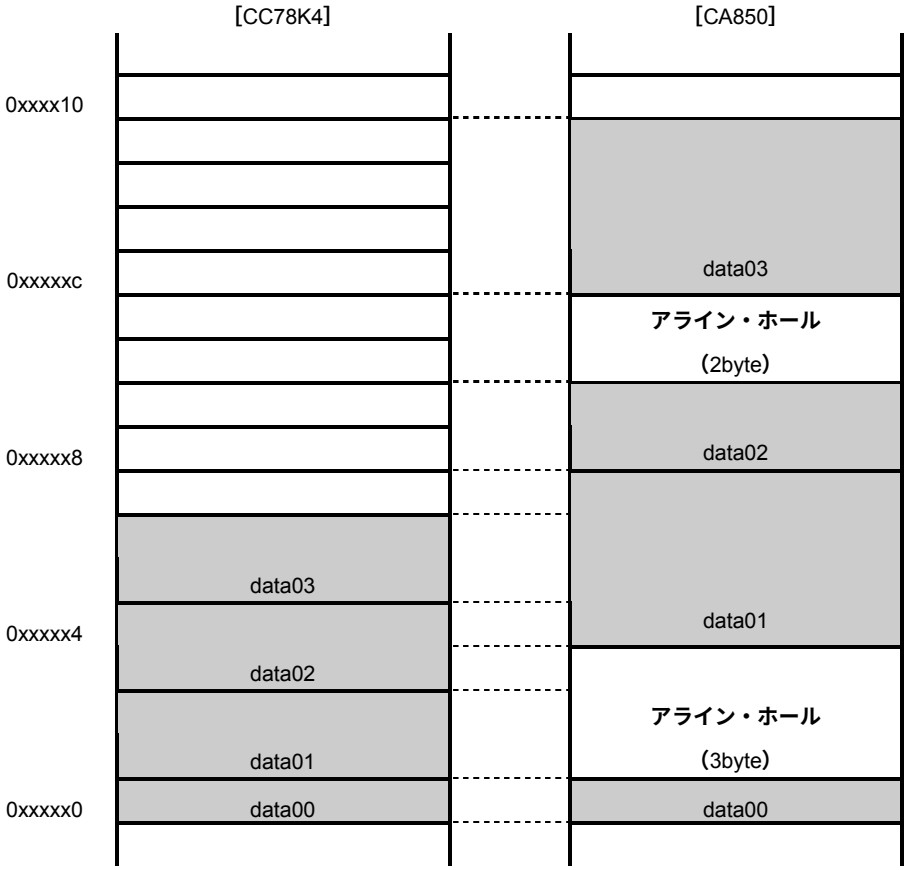
型	サイズ	整列条件
基本型	（unsigned）charとその配列	バイト境界
	（unsigned）shortとその配列	2バイト境界
	（ポインタを含む）その他の基本型	4バイト境界
共用体	2バイト＜サイズ	4バイト境界
	サイズ≦2バイト	最大メンバ・サイズ境界
構造体	2バイト＜サイズ	4バイト境界
	サイズ≦2バイト int型以上の型のメンバが存在する場合	4バイト境界
	サイズ≦2バイト int型以上のメンバがなく， 1バイト＜型のサイズ≦2バイトの場合	2バイト境界
	サイズ≦2バイト int型以上のメンバがなく， 型のサイズ≦1バイトの場合	バイト境界

＜記述例1＞

```
struct ST0 {
        char     data00;
        int      data01;
        short    data02;
        int      data03;
}ST0data;
```

RAM上のイメージは次のようになります。

図2－1　記述例1のRAMイメージ



この構造体のサイズは，アライン・ホールを含みますのでCC78K4では7バイト，CA850では16バイトになります。

また，構造体の整列条件はCC78K4では1バイト整列，CA850では4バイト整列になります。

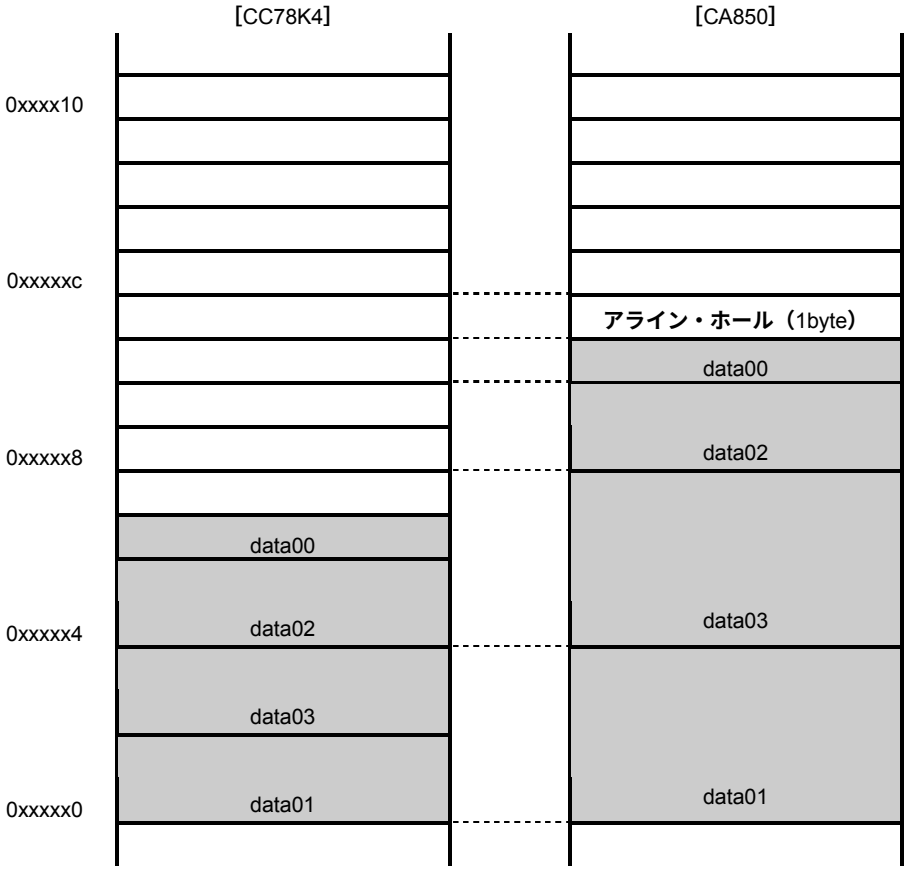
次のように，変数のサイズの大きい順に定義することにより，アライン・ホールを減らすことが可能です。

＜記述例2＞

```
struct ST0 {
        int     data01;
        int     data03;
        short   data02;
        char    data00;
}ST0data;
```

RAM上のイメージは次のようになります。

図2－2　記述例2のRAMイメージ



この構造体のサイズは，CC78K4では7バイト，CA850では12バイトになります。

2. 5　スタート・アップ・ルーチン（スタート・アップ・モジュール）

この節では，CC78K4のラージ・モデルの場合とCA850についてディフォールトのスタート・アップ・ルーチン（スタート・アップ・モジュール）の内容を機能別に比較しています。スタート・アップ・ルーチンをカスタマイズする際の参考にしてください。

また，プログラムの流れ（プログラム全体）については，各コンパイラのマニュアルを参照してください。

2.5.1　モジュール名の設定，インクルード・ファイルの取り込み

CC78K4ではモジュール名を設定し，インクルード・ファイルを取り込んでいますが，CA850にはこのような処理はありません。

【CC78K4】

```
        NAME    @cstart
$INCLUDE (mod.inc)
```

【CA850】

ありません。

2.5.2　ライブラリ・スイッチの設定

CC78K4では使用するライブラリによってスタート・アップ・ルーチンが変更できるため，使用するライブラリに対応したシンボルが定義されています。

CA850にはこのような処理はありません。

【CC78K4】

```
BRKSW    EQU    1        ;brk,sbrk,calloc,free,malloc,realloc function use
EXITSW   EQU    1        ;exit,atexit function use
RANDSW   EQU    1        ;rand,srand  function use
DIVSW    EQU    1        ;div        function use
LDIVSW   EQU    1        ;ldiv       function use
STRTOKSW EQU    1        ;strtok     function use
```

【CA850】

ありません。

2.5.3　シンボルの定義

CC78K4では，標準ライブラリ使用時に使用するシンボルがありますが，CA850にはありません。

CA850にて，V850Eを品種指定した場合に関数のプロローグ／エピローグのランタイム化をオプションで指定する場合には，シンボルを定義する必要があります。

【CC78K4】

スタート・アップ・ルーチンの開始終了シンボル

```
        PUBLIC   _@cstart,_@cend
```

標準ライブラリ使用時に使用するシンボル

```
        PUBLIC   _errno,_@BRKADR,_@MEMTOP,_@MEMBTM
        PUBLIC   _@FNCTBL,_@FNCENT,_@SEED,_@DIVR,_@LDIVR,_@TOKPTR
```

スタック解決用のシンボル

```
        EXTRN    _@STBEG
```

メイン・モジュールなどのシンボル

```
        EXTRN    _main,_hdwinit,_exit
```

ROM化処理用のシンボル

```
        EXTRN    _?R_INIT,_?R_INIS,_?R_INS1,_?DATS,_?DATS1,_?DATA
```

【CA850】

```
スタート・アップ・ルーチンの開始終了シンボル
        .globl   __start
        .globl  __exit
        .globl   __startend
リンク・エディタで自動生成されるシンボル
        .extern __tp_TEXT, 4
        .extern __gp_DATA, 4
        .extern __ep_DATA, 4
セクションの開始終了シンボル
        .extern  __ssbss, 4
        .extern __esbss, 4
        .extern __sbss, 4
        .extern __ebss, 4
メイン・モジュールのシンボル
        .extern  _main
関数のプロローグ／エピローグのランタイム化に使用するシンボル（V850Eのみ）
        .globl  ___PROLOG_TABLE
```

2. 5. 4　ライブラリ用領域の確保

CA850では，標準ライブラリの記憶域管理のライブラリを使用するときのみヒープ領域を確保する必要があります。

【CC78K4】

```
@@DATA  DSEG
_@FNCTBL:        DS       3*32
_@FNCENT:        DS       2
_@SEED:          DS       4
_@DIVR:          DS       4
_@LDIVR:         DS       8
_@TOKPTR:        DS       3
_errno:          DS       2
_@BRKADR:        DS       3
_@MEMTOP:        DS       48
_@MEMBTM:
```

【CA850】

標準ライブラリの記憶域管理のライブラリを使用する場合には，次のようにヒープ領域を設定してください。スタートアップではなく，Cソースに記述することも可能です。

```
        .sbss
        .comm    ___sysheap, HEAPSIZE, 4
        .sdata
        .globl   ___sizeof_sysheap
___sizeof_sysheap:
        .word    HEAPSIZE
```

2.5.5　スタック領域の確保

CC78K4ではリンカでスタック領域の領域を確保することが可能ですが,CA850では明示的に領域を確保する必要があります。

【CC78K4】

スタック領域の記述はありません。

リンカで-Sオプションを指定することにより，メモリ領域の中で最大のアドレス領域を探し_@STBG，_@STENDシンボルを生成します。

【CA850】

```
        .set     STACKSIZE, 0x200
        .bss
        .lcomm   __stack, STACKSIZE, 4
```

注意　スタックは必ず4バイト単位で確保してください。

2.5.6　リセット・ベクタの設定

CC78K4，CA850ともに，リセット・ベクタの設定を#pragma指令で記述することはできません。そのため，スタート・アップ・ルーチンに記述があります。

【CC78K4】

```
@@VECT00CSEG     AT       0H
        DW       _@cstart
```

【CA850】

```
        .section       "RESET", text
        jr     __start
```

2.5.7　ロケーションの設定

78K/IVシリーズでのロケーション命令は，V850ファミリにはありません。

【CC78K4】

```
        LOCATION          0FH
```

【CA850】

ありません。

2.5.8　レジスタ・バンクの設定

78K/**IV**シリーズでのレジスタ・バンクは，V850ファミリにはありません。

【CC78K4】

```
        SEL     RB0
```

【CA850】

ありません。

2.5.9　スタック・ポインタの設定

CC78K4では，リンカで生成したシンボルをスタック・ポインタに設定します。

CA850では，明示的に確保した領域の先頭のシンボルに，スタックのサイズを足したものをスタック・ポインタに設定します。

【CC78K4】

```
        MOVG    SP,#_@STBEG
```

【CA850】

```
        mov     #__stack+STACKSIZE, sp
```

2.5.10　汎用レジスタの設定

CA850では，プログラム，変数にアクセスする際に，レジスタの値からの相対番地でアクセスします。そのため，汎用レジスタの設定が必要になります。

また，マスク・レジスタ機能を使用する場合も設定が必要となります。

【CC78K4】

ラージ・モデルの場合にはありません。ミディアム・モデル，スモール・モデルの場合はサンプルを参照してください。

【CA850】

```
        mov     #__tp_TEXT, tp          -- テキスト・ポインタの設定
        mov     #__gp_DATA, gp          --
        add     tp, gp                  -- グローバル・ポインタの設定
        mov     #__ep_DATA, ep          -- エレメント・ポインタの設定
        mov     0xff,   r20             -- マスク・レジスタ（バイト）の設定
        mov     0xffff, r21             -- マスク・レジスタ（ハーフ・ワード）の設定
```

リンク・ディレクティブ・ファイルにて，“グローバル・ポインタはテキスト・ポインタからのオフセットとする”と指定があることを仮定しています。そのため，tpとgpを加算してgpに代入しています。

また，マスク・レジスタ（r20, r21）はマスク・レジスタの使用をオプションで指定している場合にのみ必要です。

2. 5. 11　特殊レジスタの設定

CA850では，V850Eで関数のプロローグ／エピローグのランタイム化を使用する場合にのみ，特殊レジスタの設定が必要になります。

【CC78K4】

ありません。

【CA850】

関数のプロローグ／エピローグのランタイム化をする場合には，CALLTテーブル・ポインタに，次の設定をしてください（V850Eのみ）。

```
        mov     #___PROLOG_TABLE, r12
        ldsr    r12, 20
```

2. 5. 12　ハードウエア・イニシャライズ関数の呼び出し

CC78K4では，ユーザがカスタマイズできるように，ハードウエア・イニシャライズ関数の呼び出しがありますが，CA850ではありません。ハードウエアを初期化する場合には，CC78K4のように関数を呼び出す形にするか，そのままスタート・アップ・ルーチンに記述してください。

【CC78K4】

```
        CALL     !!_hdwinit
```

【CA850】

ありません。

2. 5. 13　標準ライブラリ用の初期値設定

CC78K4では，標準ライブラリを使用する際にシンボルの初期値設定が必要ですが，CA850では必要ありません。

ただし，CA850にて標準ライブラリを使用する場合には，初期値あり変数が存在するためROM化処理（romp850）が必要になりますので，注意してください。

【CC78K4】

```
        SUBW    AX,AX
        MOVW    !!_errno,AX             ;errno <- 0
        MOVW    !!_@FNCENT,AX           ;FNCENT <- 0
        MOVW    !!_@SEED+2,AX
        MOVW    !!_@SEED,#1             ;SEED <- 1
        MOVG    WHL,#_@MEMTOP
        MOVG    !!_@BRKADR,WHL    ;BRKADR <- #MEMTOP
```

【CA850】

ありません。

2. 5. 14　ROM化処理

CC78K4ではROM化処理をスタート・アップ・モジュールに記述してありますが，CA850ではCソースで初めに実行される部分（main関数の先頭）でROM化ライブラリを呼び出してください。

メモリを拡張する場合には，ROM化ライブラリを呼び出す前に，必要な周辺機能レジスタの設定を行ってください。

【CC78K4】

（1/2）

```
; copy external variables having initial value
        MOVG    TDE,#_@INIT
        MOVG    WHL,#_@R_INIT
LINIT1:
        SUBG    WHL,#_?R_INIT
        BE      $LINIT2
        ADDG    WHL,#_?R_INIT
        MOV     A,[HL+]
        MOV     [DE+],A
        BR      $LINIT1
LINIT2:
; copy external variables which doesn't have initial value
        MOVG    TDE,#_@DATA
        MOVG    WHL,#_?DATA
        MOV     A,#0
LDATA1:
        SUBG    WHL,TDE
        BE      $LDATA2
        ADDG    WHL,TDE
```

(2/2)

```
        MOV     [DE+],A
        BR      $LDATA1
LDATA2:
; copy sreg variables having initial value
        MOVG    TDE,#_@INIS
        MOVG    WHL,#_@R_INIS
LINIS1:
        SUBG    WHL,#_?R_INIS
        BE      $LINIS2
        ADDG    WHL,#_?R_INIS
        MOV     A,[HL+]
        MOV     [DE+],A
        BR      $LINIS1
LINIS2:
; copy sreg variables which doesn't have initial value
        MOVG    TDE,#_@DATS
        MOVG    WHL,#_?DATS
        MOV     A,#0
LDATS1:
        SUBG    WHL,TDE
        BE      $LDATS2
        ADDG    WHL,TDE
        MOV     [DE+],A
        BR      $LDATS1
LDATS2:
; copy sreg1 variables having initial value
        MOVG    TDE,#_@INIS1
        MOVG    WHL,#_@R_INS1
LINIS11:
        SUBG    WHL,#_?R_INS1
        BE      $LINIS12
        ADDG    WHL,#_?R_INS1
        MOV     A,[HL+]
        MOV     [DE+],A
        BR      $LINIS11
LINIS12:
; copy sreg1 variables which doesn't have initial value
        MOVG    TDE,#_@DATS1
        MOVG    WHL,#_?DATS1
        MOV     A,#0
LDATS11:
        SUBG    WHL,TDE
        BE      $LDATS12
        ADDG    WHL,TDE
        MOV     [DE+],A
        BR      $LDATS11
LDATS12:
```

【CA850】

main関数の先頭でコピー・ルーチン（_rcopy）を呼び出してください。

```
extern   unsigned long     _S_romp;
main()
{
         int       ret;
         ret = _rcopy(&_S_romp,-1);
         …
}
```

2. 5. 15　初期値なし変数領域の初期化

CA850では初期値なしのセクションの開始／終了のシンボルを利用して，初期化を行っています。

【CC78K4】

ありません。

【CA850】

```
         mov      #__ssbss, r13              -- clear sbss section
         mov      #__esbss, r12
         cmp      r12, r13
         jnl      .L11
.L12:
         st.w     r0, [r13]
         add      4, r13
         cmp      r12, r13
         jl       .L12
.L11:
         mov      #__sbss, r13               -- clear bss section
         mov      #__ebss, r12
         cmp      r12, r13
         jnl      .L14
.L15:
         st.w     r0, [r13]
         add      4, r13
         cmp      r12, r13
         jl       .L15
.L14:
```

上記のシンボルはリンク・エディタで予約語とされています。

__ssbss　：.sbssセクションの開始のシンボル

__esbss　：.sbssセクションの終了のシンボル

__sbss　：.bssセクションの開始のシンボル

__ebss　：.bssセクションの終了のシンボル

2. 5. 16　main関数の呼び出し

main関数を呼び出します。

【CC78K4】

```
        CALL    !!_main
```

【CA850】

```
        jarl    _main, lp
```

2. 5. 17　exit関数の呼び出し

CC78K4では，exit関数を呼び出しますが，CA850ではhalt命令を実行しHALTモードにします。

【CC78K4】

```
        SUBW    AX,AX
        CALL    !!_exit
        BR      $$
```

【CA850】

ありません。そのままHALTモードにします。

```
        halt
```

2. 5. 18　セグメント（セクション）の定義

CC78K4では，ROM化処理で使用するセグメント，ラベルを定義しています。

CA850では，初期値なしセクションを初期化するために，初期値なしセクションを定義しています。

【CC78K4】

```
@@R_INIT CSEG
_@R_INIT:
@@R_INIS CSEG
_@R_INIS:
@@R_INS1 CSEG
_@R_INS1:
@@INIT   DSEG
_@INIT:
@@DATA   DSEG
_@DATA:
@@INIS   DSEG     SADDR2
_@INIS:
@@DATS   DSEG     SADDR2
_@DATS:
@@INIS1  DSEG     SADDR
_@INIS1:
@@DATS1  DSEG     SADDR
_@DATS1:
@@CODE   CSEG
@@CALF   CSEG     FIXED
@@CNST   CSEG
@@CALT   CSEG     CALLT0
@@BITS   BSEG     SADDR2
@@BITS1  BSEG     SADDR
```

【CA850】

```
        .sbss
        .lcomm   __sbss_dummy, 0, 0
```

2. 6　コンパイラ出力のセグメント（セクション）

この節では，CC78K4の出力するセグメントとCA850の出力するセクションについて説明します。
配置の設定方法については，第4章　リンク・ディレクティブ編を参照してください。

2. 6. 1　CC78K4の出力セグメント

CC78K4のディフォールトで出力するセグメントは，次のようになります。

表2－7　CC78K4出力セグメント

セクション名	セグメント・タイプ	用途
@@BASE	CSEG	callt関数，割り込み関数セグメント
@@VECTnn	CSEG	割込ベクタ・テーブル用セグメント
@@CODES @@CODE	CSEG	通常関数コード部用セグメント
@@CNSTS @@CNSTM @@CNST	CSEG	const変数用セグメント
@@CALFS @@CALF	CSEG	callf関数用セグメント
@@CALT	CSEG	callt関数テーブル用セグメント
@@RSINIT @@R_INIT	CSEG	初期値あり変数用セグメント
@@RSINS @@R_INS	CSEG	初期値ありsreg変数用セグメント
@@RSINS1 @@R_INS1	CSEG	初期値ありsreg1変数用セグメント
@@INITM @@INIT	DSEG	初期値あり仮変数用セグメント
@@DATAM @@DATA	DSEG	初期値なし変数用セグメント
@@INIS	DSEG	初期値あり仮sreg変数用セグメント
@@DATS	DSEG	初期値なし仮sreg変数用セグメント
@@INIS1	DSEG	初期値あり仮sreg1変数用セグメント
@@DATS1	DSEG	初期値なし仮sreg1変数用セグメント
@@BITS	BSEG	boolean / bit型変数用セグメント
@@BITS1	BSEG	__boolean1型変数用セグメント

2. 6. 2　CA850の出力セクション

CA850のディフォールトで出力するセクションは，次のようになります。

表2－8　CA850出力セクション

セクション名	セクションの種類	用途
.text	text	関数コード用セクション
.pro_epi_runtime	text	プロローグ／エピローグのランタイム・ライブラリ・コード用セクション
.const	const	const変数用セクション r0相対のld / st命令（2命令）でアクセス可能なセクション
.sconst	const	const変数用セクション r0相対のld / st命令でアクセス可能なセクション （0x0～0x8000番地）
.data	data	初期値あり変数用セクション gp相対のld / st命令（2命令）でアクセス可能なセクション
.sdata	sdata	初期値あり変数用セクション gp相対のld / st命令でアクセス可能なセクション
.sedata	sedata	初期値あり変数用セクション ep相対のld / st命令でアクセス可能なセクション （内蔵RAMの先頭-0x8000～内蔵RAMの先頭番地を推奨）
.sidata	sidata	初期値あり変数用セクション ep相対のld / stでアクセス可能なセクション （内蔵RAMを推奨）
.tidata	tidata	初期値あり変数用セクション ep相対のsld / sstでアクセス可能なセクション （内蔵RAMの先頭～256byteを推奨）
.bss	bss	初期値なし変数用セクション gp相対のld / st命令（2命令）でアクセス可能なセクション
.sbss	sbss	初期値なし変数用セクション gp相対のld / st命令でアクセス可能なセクション
.sebss	sebss	初期値なし変数用セクション ep相対のld / st命令でアクセス可能なセクション （内蔵RAMの先頭-0x8000～内蔵RAMの先頭番地を推奨）
.sibss	sibss	初期値なし変数用セクション ep相対のld / stでアクセス可能なセクション （内蔵RAMを推奨）
.tibss	tibss	初期値なし変数用セクション ep相対のsld / sstでアクセス可能なセクション （内蔵RAMの先頭～256byteを推奨）
rompsec	text	初期値ありデータのコピー用にパッキングされた初期値ありデータ・セクションと，これらのセクションのアドレス情報を含むセクション

2.7　ライブラリ，ヘッダ・ファイル

この節では，CC78K4，CA850**でサポートされているライブラリ，ヘッダ・ファイルについて説明します。**

CC78K4**でサポートされているが，**CA850**ではサポートされていないライブラリを使用している場合には，**CA850**でサポートしているライブラリに置き換えて使用してください。**

表2－9　**ライブラリ，ヘッダ・ファイル（**1/2**）**

機能概略	CC78K4	CA850
文字，文字列関数	isalnum, isalpha, iscntrl, isdigit, isgragh, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, isascii, toascii, _toupper, _tolower, tolow*, toup*	_tolower, _toupper, toascii, tolower, toupper, isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit
プログラム制御関数	setjmp, longjmp	setjmp, longjmp
特殊関数	va_start, va_arg, va_end	va_start, va_arg, va_end
入出力関数	sprintf, sscanf, printf, scanf, vprintf, vsprintf, getchar, gets, putchar, puts	sprintf, sscanf, fprintf*, fscanf*, printf, scanf, vfprintf*, vprintf, vsprintf, fgetc*, fgets*, fputc*, fputs*, getc*, getchar, gets, putc*, putchar, puts, ungetc*, fread*, fwrite*, rewind*, perror*
ユーティリティ関数	atoi, atol, strtol, strtoul, calloc, free, malloc, realloc, abort*, atexit*, exit*, abs, labs, div, ldiv, brk*, sbrk*, atof*, strtod*, itoa, ltoa, ultoa, rand, srand, bsearch, qsort, strbrk*, strsbrk*, stritoa*, strltoa*, strultoa*	abs, labs, bsearch, div, ldiv, ecvtf*, gcvtf*, atoff*, strtodf*, atoi, atol, strtol, strtoul, rand, srand, calloc, malloc, free, realloc, qsort, itoa, ltoa, ultoa
文字列／メモリ関数	memcpy, memmove, strcpy, strncpy, strcat, strncat, memcmp, strcmp, strncmp, memchr, strchr, strrchr, strspn, strcspn, strpbrk, strstr, strtok, memset, strerror, strlen, strcoll, strxfrm	bcmp*, bcopy*, memchr, memcmp, memcpy, memmove, memset, index*, rindex*, strcat, strchr, strcmp, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, strtok
数学関数	acos*, asin*, atan*, atan2*, cos*, sin*, tan*, cosh*, sinh*, tanh*, exp*, frexp*, ldexp*, log*, log10*, modf*, pow*, sqrt*, ceil*, fabs*, floor*, fmod*, matherr, acosf, asinf, atanf, atan2f, cosf, sinf, tanf, coshf, sinhf, tanhf, expf, frexpf, ldexpf, logf, log10f, modff, powf, sqrtf, ceilf, fabsf, floorf, fmodf, __assertfail*	j0f*, j1f*, jnf*, y0f*, y1f*, ynf*, erff*, erfcf*, expf, logf, log2f*, log10f, powf, sqrtf, ceilf, fabsf, floorf, fmodf, frexpf, ldexpf, modff, gammaf*, hypotf*, matherr, acoshf*, asinhf*, atanhf*, coshf, sinhf, tanhf, acosf, asinf, atanf, atan2f, cosf, sinf, tanf, cbrtf*

表2－9　ライブラリ，ヘッダ・ファイル（2/2）

機能概略	CC78K4	CA850
整数演算[注]	lsinc*, luinc*, lsdec*, ludec*, lsrev*, lurev*, lscom*, lucom*, lsnot*, lunot*, lsmul*, lumul*, csdiv*, isdiv*, lsdiv*, ludiv*, csrem*, isrem*, lsrem*, lurem*, lsadd*, luadd*, lssub*, lusub*, lslsh*, lulsh*, lsrsh*, lursh*, lscmp*, lucmp*, lsband*, luband*, lsbor*, lubor*, lsbxor*, lubxor*	__mul*, __mulu*, __div*, __divu*, __mod*, __modu*
浮動小数点演算[注]	finc*, fdec*, frev*, fnot*, fmul*, fdiv*, fadd*, fsub*, fcmp*, fand*, for*, ftols*, ftoul*, lstof*, lutof*, btol*,	__addf.s*, __subf.s*, __mulf.s*, __divf.s*, __cmpf.s*, __cvt.ws*, __trnc.sw*
ROM**化用** **初期値データのコピー**	**なし**	_rcopy*
ヘッダ・ファイル	ctype.h, setjmp.h, stdarg.h, stdio.h, stdlib.h, string.h, error.h*, errno.h, limits.h, stddef.h, math.h, float.h, assert.h*	ctype.h, setjmp.h, stdarg.h, stdio.h, stdlib.h, string.h, errno.h, limits.h, stddef.h, math.h, float.h

注　ランタイム・ライブラリなので，Cソース・プログラム中には記述しません。

表2－9で，“*”がついているものがCC78K4**，**CA850**のいずれかでしかサポートされていないものです。**
ライブラリ詳細については，各コンパイラのマニュアルを参照してください。

第 3 章　アセンブリ言語編

この章では，アセンブラの疑似命令，制御命令を RA78K4 から CA850（as850）に置き換える場合に注意する点，および基本的な記述方法について説明します。

RA78K4，CA850 の疑似命令，制御命令は次の表のようになります。

表3－1　疑似命令，制御命令（1/3）

項番	RA78K4		CA850	
	機能	命令	機能	命令
1	セグメント疑似命令	CSEG	セクション定義疑似命令	.text .const .sconst
		DSEG		.bss .data .sbss .sdata .sebss .sedata .sibss .sidata .tibss .tidata .tibss.byte .tidata.byte .tibss.word .tidata.word
		BSEG		なし
		なし		.previous .section .vdbstrtab .vdebug .vline
		ORG	ロケーション・カウンタ制御疑似命令	.org
		なし		.align
2	シンボル定義疑似命令	EQU	なし	
		SET	シンボル制御疑似命令	.set
		なし		.size .frame .file
3	オブジェクト・モジュール名宣言疑似命令	NAME	なし	

表3－1　疑似命令，制御命令（2/3）

項番	RA78K4		CA850	
	機能	命令	機能	命令
4	メモリ初期化，領域確保疑似命令	DB	領域確保疑似命令	.byte
		DW		.hword
		DG		なし
		なし		.word
		DS		.lcomm
		なし		.space
		DBIT		なし
		なし		.float
5	リンケージ疑似命令	PUBLIC	プログラム・リンケージ疑似命令	.globl
		EXTRN		.extern
		EXTBIT		なし
		なし		.comm
6	自動選択疑似命令	BR CALL	なし	
7	汎用レジスタ選択疑似命令	RSS	なし	
8	マクロ疑似命令	MACRO	マクロ疑似命令	.macro
		LOCAL		.local
		ENDM		.endm
		EXITM	スキップ疑似命令	.exitm
		なし		.exitma
		REPT	繰り返しアセンブル疑似命令	.repeat
		IRP		.irepeat
9	アセンブル終了疑似命令	END	なし	
10	アセンブル対象品種指定制御命令	$PROCESSOR	アセンブラ制御疑似命令	.option cpu
11	ディバグ情報出力制御命令	$DEBUG/ $NODEBUG $DEBUGA/ $NODEBUGA	なし	
12	クロス・レファレンス・リスト出力指定制御命令	$XREF/$NOXREF $SYMLIST/ $NOSYMLIST	なし	
13	インクルード制御命令	$INCLUDE	ファイル入力制御疑似命令	.include
		なし		.binclude

表3－1　疑似命令，制御命令（3/3）

項番	RA78K4		CA850	
	機能	命令	機能	命令
14	アセンブル・リスト制御命令	$EJECT $TITLE $SUBTITLE $LIST/$NOLIST $GEN/$NOGEN $COND/$NOCOND $FORMFEED/ $NOFORMFEED $WIDTH $LENGTH $TAB	なし	
15	条件付きアセンブル制御命令	$SET $RESET	なし	
		$IF	条件アセンブル疑似命令	.ifdef
		なし		.ifndef
		$_IF		.if
		なし		.ifn
		$ELSEIF $_ELSEIF		.elseif
		なし		.elseifn
		$ELSE		.else
		$ENDIF		.endif
16	SFR 領域変更制御命令	$CHGSFR $CHGSFRA	なし	
17	漢字コード指定制御命令	$KANJICODE	なし	

3. 1　セグメント疑似命令（セクション定義疑似命令）

この節では，セグメント（セクション）に関する疑似命令の説明をします。

3. 1. 1　CSEG, .text, .const, .sconst

ROM に配置するセグメント（セクション）を定義する疑似命令です。

RA78K4 では，プログラムと参照のみの変数のセグメントを CSEG 疑似命令で定義しますが，CA850 では，プログラムのセクションは.text 疑似命令で，参照のみの変数のセクションは.sconst 疑似命令または.const 疑似命令で定義します。

【RA78K4】

CSEG　：コード・セグメントの開始を指示します。

記述形式は次のとおりです。

［セグメント名］　CSEG　［再配置属性］

＜記述例＞

```
        CSEG    UNIT
        MOV     A,B
        CSEG    BASE
DATAC0: DB      12H
DATAC1: DB      34H
        CSEG
DATAC2: DW      5678H
```

【CA850】

.text　　：生成するコードを.text セクションに割り当てます。
記述形式は次のとおりです。

.text

.const　　：生成するコードを.const セクションに割り当てます。
.const セクションは，定数データ用（読み出し専用）のセクションで r0 と 32 ビットのディスプレースメントを用いて 2 命令で参照されるメモリに配置されるセクションです。
記述形式は次のとおりです。

.const

.sconst　　：生成するコードを.sconst セクションに割り当てます。
.sconst セクションは，定数データ用（読み出し専用）のセクションで r0 と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からプラス方向に最大 32K バイト）に配置されるセクションです。
記述形式は次のとおりです。

.sconst

＜記述例＞

```
        .text
        .align 4
        mov     r10,r11
        .const
DATAC0:
        .byte   0x12
DATAC1:
        .byte   0x34
        .sconst
DATAC2:
        .hword  0x5678
```

3. 1. 2　DSEG, .bss, .data, .sbss, .sdata, .sebss, .sedata, .sibss, .sidata, .tibss, .tidata, .tibss.byte, .tidata.byte, .tibss.word, .tidata.word

RAM に配置するセグメント（セクション）を定義する疑似命令です。

RA78K4 では，ビット変数以外の変数のセグメントを DSEG 疑似命令で定義しますが，CA850 では，ビット変数のセクションはなく，変数のセクションは，配置するアドレスに依存しない.bss 疑似命令，.data 疑似命令で，GP レジスタ値に依存する.sbss 疑似命令または.sdata 疑似命令で定義します。また，内蔵 RAM の直前の外部の RAM には.sebss 疑似命令または.sedata 疑似命令で定義します。また，内蔵 RAM には.sibss 疑似命令，.sidata 疑似命令，.tibss 疑似命令，.tidata 疑似命令，.tibss.byte 疑似命令，.tidata.byte 疑似命令，.tibss.word 疑似命令または.tidata.word 疑似命令で定義します。

【RA78K4】

DSEG　：データ・セグメントの開始を指示します。
記述形式は次のとおりです。

［セグメント名］　DSEG　［再配置属性］

＜記述例＞

```
        DSEG    UNIT
DATAD0: DS      1
        DSEG    UNITP
DATAD2: DS      2
        DSEG
DATAD4: DS      2
        DSEG    SADDR
DATAD6: DS      2
        DSEG    SADDR2
DATAD8: DS      2
        DSEG    SADDR2
DATAD10:DS      1
        DSEG    SADDRP2
DATAD12:DS      2
```

【CA850】

.bss　：生成するコードを.bss セクションに割り当てます。
.bss セクションは，初期値を持たず，gp と 32 ビットのディスプレースメントを用いて 2 命令で参照されるメモリ範囲に配置されるセクションです。
記述形式は次のとおりです。

.bss

.data　：生成するコードを.data セクションに割り当てます。
.data セクションは，初期値を持ち，gp と 32 ビットのディスプレースメントを用いて 2 命令で参照されるメモリ範囲に配置されるセクションです。
記述形式は次のとおりです。

```
.data
```

.sbss　：生成するコードを.sbss セクションに割り当てます。
.sbss セクションは，初期値を持たず，gp と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（.sdata セクションとあわせて最大 64K バイト）に配置されるセクションです。
記述形式は次のとおりです。

```
.sbss
```

.sdata　：生成するコードを.sdata セクションに割り当てます。
.sdata セクションは，初期値を持ち，gp と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（.sbss セクションとあわせて最大 64K バイト）に配置されるセクションです。
記述形式は次のとおりです。

```
.sdata
```

.sebss　：生成するコードを.sebss セクションに割り当てます。
.sebss セクションは，初期値を持たず，ep と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からマイナス方向に 32K バイト）のうち，.sedata セクションのサイズ分だけ上位に配置されるセクションです。
記述形式は次のとおりです。

```
.sebss
```

.sedata　：生成するコードを.sedata セクションに割り当てます。
.sedata セクションは，初期値を持ち，ep と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からマイナス方向に 32K バイト）のうち，.sebss セクションのサイズ分だけ下位に配置されるセクションです。
記述形式は次のとおりです。

```
.sedata
```

.sibss　：生成するコードを.sibss セクションに割り当てます。

.sibss セクションは，初期値を持たず，ep と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に 32K バイト，つまり，内蔵 RAM）に配置されるセクションです。

記述形式は次のとおりです。

.sibss

.sidata　：生成するコードを.sidata セクションに割り当てます。

.sidata セクションは，初期値を持ち，ep と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に 32K バイト，つまり，内蔵 RAM）に配置されるセクションです。

記述形式は次のとおりです。

.sidata

.tibss　：生成するコードを.tibss セクションに割り当てます。

.tibss セクションは，sld/sst 命令[注]を用いて ep 相対でアクセスされることを前提にした初期値なしのデータのセクションです。.tibss.byte，.tibss.word の両セクションが使用されている場合は，.tibss を ep の示すアドレスに両セクションのサイズを加えたアドレスへ配置します。

記述形式は次のとおりです。

.tibss

.tidata　：生成するコードを.tidata セクションに割り当てます。

.tidata セクションは，sld/sst 命令[注]を用いて ep 相対でアクセスされることを前提にした初期値を持つデータのセクションです。.tidata.byte，.tidata.word の両セクションが使用されている場合は，.tidata を ep の示すアドレスに両セクションのサイズを加えたアドレスへ配置します。

記述形式は次のとおりです。

.tidata

注　sld/sst命令とは，アクセスするデータがバイト・データの場合128バイト以内，ハーフ・ワード以上のデータの場合256バイト以内の領域をアクセスできる2バイトの命令です。

.tibss.byte　　：生成するコードを.tibss.byte セクションに割り当てます。

.tibss.byte セクションは，sld/sst 命令[注]を用いて ep 相対でアクセスされることを前提にした初期値なしのデータのセクションです。sld/sst 命令[注]がアクセスできる領域を有効に使用するため，データのサイズによってセクションを.tibss.byte セクションと.tibss.word セクションにわけ.tibss.byte セクションを ep が示すアドレスに配置するようにしています。

記述形式は次のとおりです。

.tibss.byte

.tidata.byte　　：生成するコードを.tidata.byte セクションに割り当てます。

.tidata.byte セクションは，sld/sst 命令[注]を用いて ep 相対でアクセスされることを前提にした初期値を持つデータのセクションです。sld/sst 命令[注]がアクセスできる領域を有効に使用するため，データのサイズによってセクションを.tidata.byte セクションと.tidata.word セクションにわけ.tidata.byte セクションを ep が示すアドレスに配置するようにしています。

記述形式は次のとおりです。

.tidata.byte

.tibss.word　　：生成するコードを.tibss.word セクションに割り当てます。

.tibss.word セクションは，sld/sst 命令[注]を用いて ep 相対でアクセスされることを前提にした初期値なしのデータのセクションです。sld/sst 命令[注]がアクセスできる領域を有効に使用するため，データのサイズによってセクションを.tibss.byte セクションと.tibss.word セクションにわけ.tibss.word セクションを ep が示すアドレスに.tibss.byte セクションのサイズを加えたアドレスに配置するようにしています。

記述形式は次のとおりです。

.tibss.word

注　sld/sst命令とは，アクセスするデータがバイト・データの場合128バイト以内，ハーフ・ワード以上のデータの場合256バイト以内の領域をアクセスできる2バイトの命令です。

.tidata.word　　：生成するコードを.tidata.word セクションに割り当てます。

.tidata.word セクションは，sld/sst 命令[注]を用いて ep 相対でアクセスされることを前提にした初期値を持つデータのセクションです。sld/sst 命令[注]がアクセスできる領域を有効に使用するため，データのサイズによってセクションを.tidata.byte セクションと.tidata.word セクションにわけ.tidata.word セクションを ep が示すアドレスに.tidata.byte セクションのサイズを加えたアドレスに配置するようにしています。

記述形式は次のとおりです。

.tidata.word

注　sld/sst命令とは，アクセスするデータがバイト・データの場合128バイト以内，ハーフ・ワード以上のデータの場合256バイト以内の領域をアクセスできる2バイトの命令です。

＜記述例＞

```
        .bss
        .lcomm   DATAD0,0x1,1
        .data
DATAD1:
        .byte    0xff
        .sbss
        .lcomm   DATAD2,0x2,2
        .sdata
DATAD3:
        .hword   0xffff
        .sebss
        .lcomm   DATAD4,0x2,2
        .sedata
DATAD5:
        .hword   0xabcd
        .sibss
        .lcomm   DATAD6,0x2,2
        .sidata
DATAD7:
        .hword   0xfedc
        .tibss
        .lcomm   DATAD8,0x2,2
        .tidata
DATAD9:
        .hword   0x4321
        .tibss.byte
        .lcomm   DATAD10,0x1,1
        .tidata.byte
DATAD11:
        .byte    0x21
        .tibss.word
        .lcomm   DATAD12,0x2,2
        .tidata.word
DATAD13:
        .hword   0x5678
```

3. 1. 3　BSEG

RAM に配置するセグメント（セクション）を定義する疑似命令です。

RA78K4 では，ビット変数のセグメントを BSEG 疑似命令で定義しますが，CA850 では，ビット変数のセクションはありません。

【RA78K4】

BSEG　：ビット・セグメントの開始を指示します。

記述形式は次のとおりです。

［セグメント名］　BSEG　［再配置属性］

＜記述例＞

```
        BSEG
DATAB0  DBIT
DATAB1  DBIT
        CSEG
        SET1    DATAB0
        CLR1    DATAB1
```

【CA850】

ありません。

バイトなどのサイズで領域を確保し，set1，clr1，tst1 命令でビット位置を指定してアクセスしてください。

＜記述例＞

```
        .bss
        .lcomm  DATAB,0x1,1
        .text
        set1    0,$DATAB[gp]
        clr1    1,$DATAB[gp]
```

3. 1. 4　.previous, .section

セグメント（セクション）を定義する疑似命令です。

RA78K4 **では，セグメント名を指定する場合には，**CSEG **疑似命令，**DSEG **疑似命令，**BSEG **疑似命令で名前を指定することが可能ですが，**CA850 **では，**.section **疑似命令でセクション名を指定します。また，**CA850 **では**.previous **で前のセクション疑似命令で指定したものを再指定することが可能です。**

【RA78K4】

.previous **に相当する疑似命令はありません。**

セグメント名は CSEG**，**DSEG**，**BSEG **にて指定することが可能です。**

＜記述例＞

```
SEG1    CSEG
        MOV     A,B
SEG2    DSEG
SEGL0:  DS      2
```

【CA850】

.previous　**：現在のセクション疑似命令を指定しているセクション定義疑似命令の前のセクション定義疑似命令を（再）指定します。**
記述形式は次のとおりです。

. previous

.section　**：プログラムに対して生成されるコードを第一オペランドに指定したセクション名で，第二オペランドに指定した種類のセクションに割り付けます。**
記述形式は次のとおりです。

. section "**セクション名**"［,**セクションの種類**］

＜記述例＞

```
.section "SEG1",text
mov     r10,r11
.section "SEG2",bss
.lcomm  SEGL0,2,2
```

3. 1. 5　ORG, .org

セグメント（セクション）を定義する疑似命令です。

RA78K4 **では，セグメントの絶対番地を指定する場合に** ORG **疑似命令を使用しますが，**CA850 **では，セクションの絶対番地を指定することはできずに，現在のセクションのロケーション・カウンタを進めるだけです。絶対番地を指定する場合には，リンク・ディレクティブ・ファイルでセクションの配置を指定してください。**

【RA78K4】

ORG　　**：ロケーション・カウンタにオペランドで指定した式の値を設定します。**
ORG 疑似命令後はアブソリュート・セグメントに属しオペランドで指定した
アドレスから配置されます。
記述形式は次のとおりです。

［セグメント名］　ORG　**絶対式**

＜記述例＞

” MOV　A,C**”は，**0x100 **番地に配置されるセグメントに配置されます。**

```
        CSEG
        MOV     A,B
        ORG     0100H
        MOV     A,C
```

【CA850】

.org　　**：前に指定されたセクション定義疑似命令によって指定される現在のセクションに対するロケーション・カウンタ値をオペランドで指定した値まで進めます。**
セクションの絶対番地を指定するものではありません。なお，ロケーション・カウンタ値を進めることによりホールが生じた場合，生じたホールを 0 **で埋めます。**
記述形式は次のとおりです。

.org　**値**

＜記述例＞

“mov　r12,r11**”は**.text **セクションのロケーション・カウンタを** 0x100 **に進めたところに配置されます。**

```
        .text
        mov     r10,r11
        .org    0x100
        mov     r12,r11
```

3. 1. 6　.align

セグメント（セクション）の整列条件を指定する疑似命令です。

RA78K4 **ではありませんが，**CA850 **では，**.align **疑似命令でセクションの整列条件を指定することが可能です。**

【RA78K4】

ありません。

【CA850】

.align　**：前に指定されたセクション定義疑似命令によって指定される現在のセクションに対するロケーション・カウンタ値を第一オペランドで指定した整列条件で整列します。**
なお，ロケーション・カウンタ値を整列したことによりホールが生じた場合，生じたホールを第二オペランドで指定したフィリング値またはディフォールト値の 0 **で埋めます。**
記述形式は次のとおりです。

.align　**整列条件［,フィリング値］**

＜記述例＞

```
        .text
        .align 4
        .globl   func
func:
        jmp      [lp]
```

3. 2　シンボル定義疑似命令（シンボル制御疑似命令）

この節では，シンボルに関する疑似命令の説明をします。

3. 2. 1　EQU

RA78K4 では，ネームを定義する EQU 疑似命令と SET 疑似命令がありますが，CA850 では，ネームに値を設定する.set 疑似命令のみのサポートになります。

【RA78K4】

EQU　　：オペランドで指定した式の値を持つネームを定義します。
記述形式は次のとおりです。

ネーム　EQU　**式**

＜記述例＞

```
DATAE       EQU     0FE00H
```

【CA850】

ありません。
数値の場合には，.set 疑似命令を使用してください。

3. 2. 2　SET, .set

RA78K4 では，ネームを定義する場合 SET 疑似命令を使用しますが，CA850 では，ネームに値を設定する.set 疑似命令を使用します。

【RA78K4】

SET　　：オペランドで指定した式の値を持つネームを定義します。
同一モジュール内において再定義が可能です。
記述形式は次のとおりです。

ネーム　SET　**絶対式**

＜記述例＞

```
DATAS       SET     100
```

【CA850】

.set　　：第一オペランドに指定したシンボル名と第二オペランドに指定した値（整数値）を持つシンボルを定義します。
記述形式は次のとおりです。

.set　**シンボル名，値**

＜記述例＞

```
.set        DATAE,0xfe00
```

3. 2. 3　.size, .frame, .file

CA850 では，ラベルにサイズを持たせる.size 疑似命令，C 言語レベルでのディバグのための.frame 疑似命令，ファイル名を定義する.file 疑似命令があります。

【RA78K4】

ありません。

【CA850】

.size　：第二オペランドに指定したサイズを，第一オペランドに指定したラベルの示すデータのサイズとして指定します。
記述形式は次のとおりです。

.size　ラベル名，サイズ

.frame　：オブジェクト・ファイル生成時，第一オペランドに指定したラベルに対するシンボル・テーブル・エントリの生成において，第二オペランドに指定したサイズとタイプ FUNC を持つシンボル・テーブル・エントリを生成します。
C 言語レベルでのディバグのために使用する疑似命令です。
記述形式は次のとおりです。

.frame　ラベル名，サイズ

.file　：オブジェクト・ファイルの生成時，オペランドに指定したファイル名と，タイプ FUNC を持つシンボル・テーブル・エントリを生成します。
記述形式は次のとおりです。

.file　“ファイル名”

＜記述例＞

```
    .file       "main.s"
```

3. 3　オブジェクト・モジュール名宣言疑似命令

この節では，オブジェクト・モジュール名に関する疑似命令の説明をします。

3. 3. 1　NAME

RA78K4 では，オブジェクト・モジュールの名前を定義する NAME 疑似命令がありますが，CA850 には同等の疑似命令はありません。

【RA78K4】

NAME　：オペランドに記述したオブジェクト・モジュール名を，アセンブラの出力するオブジェクト・モジュールに与えます。
記述形式は次のとおりです。

NAME　オブジェクト・モジュール名

【CA850】

ありません。

3. 4　メモリ初期化，領域確保疑似命令（領域確保疑似命令）

この節では，メモリの初期化，メモリの領域確保する疑似命令について説明をします。

3. 4. 1　DB, .byte

メモリの 1 バイトの領域を確保する場合に，RA78K4 では DB 疑似命令を，CA850 では.byte 疑似命令を使用します。

【RA78K4】

DB：バイト領域（1 バイト）を初期化します。
オペランドにサイズを指定した場合には，指定されたサイズ分の領域を 0 で初期化します。
オペランドに初期値を指定した場合は，指定された初期値で初期化します。
記述形式は次の 2 通りがあります。

DB　（サイズ）
DB　初期値［，…］

＜記述例＞

```
        CSEG
DBDATA0:DB      (1)
DBDATA1:DB      0AAH,0BBH
```

【CA850】

.byte　：最初の形式は，各オペランドに対して 1 バイト分の領域を確保し，確保した領域に指定した値の下位 1 バイトの値を格納します。
2 番目の形式は，指定したビット幅の領域を確保し，確保した領域に指定した値を格納します。
記述形式は次の 2 通りがあります。

.byte　**値［,値…］**

.byte　**ビット幅：値［,ビット幅:値…］**

＜記述例＞

```
        .sdata
DBDATA0:
        .space   1
DBDATA1:
        .byte    0xaa,0xbb
```

3. 4. 2　DW, .hword

メモリの 2 バイトの領域を確保する場合に，RA78K4 **では** DW **疑似命令を，**CA850 **では**.hword **疑似命令を使用します。**

【RA78K4】

DW　：ワード領域（2 バイト）を初期化します。
オペランドにサイズを指定した場合には，指定されたサイズ×2 バイト分の領域を 0 で初期化します。
オペランドに初期値を指定した場合は，指定された初期値で初期化します。
記述形式は次の 2 通りがあります。

DW　(**サイズ**)

DW　**初期値**［,…］

＜記述例＞

```
        CSEG
DWDATA0:DW       (1)
DWDATA1:DW       0CCCCH,0DDDDH
```

【CA850】

.hword　：最初の形式は，各オペランドに対して 1 ハーフ・バイト分の領域を確保し，確保した領域に指定した値の下位 1 ハーフ・バイトの値を格納します。
2 番目の形式は，指定したビット幅の領域を確保し，確保した領域に指定した値を格納します。
記述形式は次の 2 通りがあります。

.hword　値［，値…］
.hword　ビット幅：値［，ビット幅：値…］

＜記述例＞

```
        .sdata
DWDATA0:
        .space   2
DWDATA1:
        .hword   0xcccc,0xdddd
```

3. 4. 3　DG

メモリの 3 バイトの領域を確保する場合に，RA78K4 では DG 疑似命令を使用します。CA850 では 3 バイトの領域を確保する疑似命令はなく，4 バイトの領域を確保する.word 疑似命令のみサポートされています。.word 疑似命令の詳細は，3.4.4　.word を参照してください。

【RA78K4】

DG　：3 バイト領域を初期化します。
オペランドにサイズを指定した場合には，指定されたサイズ×3 バイト分の領域を 0 で初期化します。
オペランドに初期値を指定した場合は，指定された初期値で初期化します。
記述形式は次の 2 通りがあります。

DG　(サイズ)
DG　初期値［,…］

＜記述例＞

```
        CSEG
DGDATA0:DG       (1)
DGDATA1:DG       0123456H,0789ABCH
```

【CA850】

ありません。

3. 4. 4　.word

メモリの 4 **バイトの領域を確保する場合に，**CA850 **では**.word **疑似命令を使用します。**

【RA78K4】

ありません。

【CA850】

.word　**：最初の形式は，各オペランドに対して** 1 **ワード分の領域を確保し，確保した領域に指定した値の下位** 1 **ワードの値を格納します。**

2 **番目の形式は，指定したビット幅の領域を確保し，確保した領域に指定した値を格納します。**

記述形式は次の 2 **通りがあります。**

.word　**値［,値…］**

.word　**ビット幅:値［,ビット幅:値…］**

＜記述例＞

```
        .sdata
DGDATA0:
        .space   4
DGDATA1:
        .word    0x123456,0x789abc
```

3. 4. 5　.space

メモリをサイズ分確保して，確保した領域に指定値で埋める場合に，CA850 **では**.space **疑似命令を使用します。**0 **で埋める場合には，**RA78K4 **で** DB，DW，DG **にサイズを指定した場合に相当します。**

【RA78K4】

0 **で埋める場合には，**DB，DW，DG **にサイズを指定した場合に相当します。**

記述形式，例は，各疑似命令を参照してください。

【CA850】

.space　：第一オペランドで指定したサイズ分の領域を確保し，確保した領域を第二オペランドで指定したフィリング値（ディフォールト値は 0）で埋めます。
記述形式は次のとおりです。例は DB，DW，DG を参照してください。

.space　サイズ［,フィリング値］

＜記述例＞

```
        .sdata
SPDATA:
        .space   4
```

3. 4. 6　.shword

CA850 での 2 バイトの領域を確保し，確保した領域に指定した値を右 1 ビット・シフトして格納する疑似命令です。

V850E のみサポートされていて，swich 命令の際に使用します。

【RA78K4】

ありません。

【CA850】　（V850Eのみ）

.shword　：最初の形式は，各オペランドに対して 1 ハーフ・バイト分の領域を確保し，確保した領域に指定した値を右 1 ビット・シフトして格納します。
2 番目の形式は，指定したビット幅の領域を確保し，確保した領域に指定した値を右 1 ビット・シフトして格納します。
記述形式は次の 2 通りがあります。

.shword　値［,値…］

.shword　ビット幅:値［,ビット幅:値…］

＜記述例＞

0x4444 が右 1 ビット・シフトされた値の 0x2222 が領域に格納されます。

```
        .sdata
SHDATA0:
        .shword  0x4444
```

3. 4. 7　DS, .lcomm

指定したバイト分のメモリを確保する際に，RA78K4 **では** DS **疑似命令を，**CA850 **では**.lcomm **疑似命令を使用します。**

【RA78K4】

DS　　**：オペランドで指定したバイト数分のメモリ領域を確保します。**
記述形式は次のとおりです。

DS　**絶対式**

＜記述例＞

```
        DSEG
DSDATA0:DS      1
DSDATA1:DS      2
```

【CA850】

.lcomm　**：現在のセクションに対するロケーション・カウンタを第三オペランドで指定した整列条件で整列し，第二オペランドで指定したサイズの領域を確保し，その先頭のアドレスに対して第一オペランドで指定したラベル名を持つローカルなラベルを定義します。**
記述形式は次のとおりです。

.lcomm　**ラベル名**,**サイズ**,**整列条件**

＜記述例＞

```
        .sbss
        .lcomm  DSDATA0,1,1
        .lcomm  DSDATA1,2,2
```

3. 4. 8　DBIT

RA78K4 では，1 ビットのメモリを確保を DBIT 疑似命令で定義します。

CA850 では 1 ビットのメモリを確保する疑似命令はありません。バイトなどのサイズで領域を確保し set1，clr1，tst1 命令でビット位置を指定してアクセスしてください。

【RA78K4】

DBIT　　：ビット・セグメント中で 1 ビットのメモリ領域を確保します。
記述形式は次のとおりです。

ネーム　DBIT

＜記述例＞

```
        BSEG
DATAB0  DBIT
DATAB1  DBIT

        CSEG
        SET1    DATAB0
        CLR1    DATAB1
```

【CA850】

ありません。

バイトなどのサイズで領域を確保し，set1，clr1，tst1 命令でビット位置を指定してアクセスしてください。

＜記述例＞

```
        .bss
        .lcomm  DATAB,0x1,1

        .text
        set1    0,$DATAB[gp]
        clr1    1,$DATAB[gp]
```

3. 4. 9　.float, .str

CA850 では，浮動小数点値を確保する場合に.float 疑似命令を，文字列を確保する場合に.str 疑似命令を使用します。RA78K4 ではこれに相当する疑似命令はありません。

【RA78K4】

ありません。

【CA850】

.float　　：各オペランドに対して 1 ワード分の領域を確保し，確保した領域に浮動小数点値を格納します。
記述形式は次のとおりです。

.float　値［,値　…］

.str　　：各オペランドに対して，指定された文字列分の領域を確保し，確保した領域に指定した文字列を格納します。
記述形式は次のとおりです。

.str　文字列定数［,文字列定数　…］

＜記述例＞

```
        .sdata
FDATA:
        .float   1.234
STRDATA:
        .str     "NEC"
```

3. 5　リンケージ疑似命令（プログラム・リンケージ疑似命令）

この節では，リンケージに関する疑似命令について説明をします。

3. 5. 1　PUBLIC, .globl

シンボルがグローバルなシンボルであることを定義する場合に，RA78K4 では PUBLIC 疑似命令，CA850 では.globl 疑似命令を使用します。

【RA78K4】

PUBLIC　：オペランドに記述したシンボルを他のモジュールから参照できるように宣言します。
記述形式は次のとおりです。

PUBLIC　シンボル名［,シンボル名　…］

＜記述例＞

```
PUBLIC  PUBSYM
```

【CA850】

.globl　：第一オペランドで指定したラベル名と同名のラベル名を外部ラベルとして宣言します。第二オペランドを指定した場合，指定した値をそのラベルの示すデータのサイズとして指定します。

記述形式は次のとおりです。

.globl　ラベル名［,サイズ］

＜記述例＞

```
        .globl   PUBSYM
```

3. 5. 2　EXTRN, .extern

シンボルが他のモジュールで定義されているグローバルなシンボルであることを宣言する場合に，RA78K4 では EXTRN 疑似命令，CA850 では.extern 疑似命令を使用します。

【RA78K4】

EXTRN　：このモジュールで参照する他のモジュールのシンボル（ビット・シンボルを除く）を宣言します。

記述形式は次の 3 通りがあります。

EXTRN　シンボル名［,シンボル名　…］

EXTRN　SADDR2（シンボル名［,シンボル名　…］）

EXTRN　BASE（シンボル名［,シンボル名　…］）

＜記述例＞

```
EXTRN   EXSYM0
EXTRN   SADDR2(EXSYM1)
```

【CA850】

.extern　：第一オペランドで指定したラベル名と同名のラベル名を外部ラベルとして宣言します。第二オペランドを指定した場合，指定した値をそのラベルの示すデータのサイズとして指定します。

記述形式は次のとおりです。

.extern　ラベル名［,サイズ］

＜記述例＞

```
        .extern  EXSYM0
        .extern  EXSYM1
```

3. 5. 3　EXTBIT

シンボルが他のモジュールで定義されているグローバルなビット・シンボルであることを宣言する場合に，RA78K4 **では** EXTBIT **疑似命令を使用します。**CA850 **ではビット・シンボルがありませんので，この疑似命令に相当するものはありません。**

【RA78K4】

EXTBIT　**：このモジュールで参照する他のモジュールのビット・シンボルを宣言します。**
記述形式は次の 2 **通りがあります。**

EXTBIT　**シンボル名**［,**シンボル名　…**］
EXTBIT　SADDR2（**シンボル名**［,**シンボル名　…**］）

＜記述例＞

```
EXTBIT  EXBSYM0
EXTBIT  SADDR2(EXBSYM1)
```

【CA850】

ありません。
.extern **疑似命令で** 1 **バイト以上の領域を指定し，参照してください。**

3. 5. 4　.comm

CA850 **では未定義外部ラベルを定義する場合に，**.comm **疑似命令を使用します。**
RA78K4 **に，この疑似命令に相当するものはありません。**

【RA78K4】

ありません。

【CA850】

.comm　**：第一オペランドで指定したラベル名，第二オペランドで指定したサイズ，および第三オペランドで指定した整列条件を持つ未定義外部ラベルを宣言します。**
記述形式は次のとおりです。

.comm　**ラベル名，サイズ，整列条件**

＜記述例＞

```
        .comm    EXSYM1,1,1
```

3. 6　自動選択疑似命令

この節では，自動選択疑似命令について説明をします。

3. 6. 1　BR, CALL

RA78K4 では，分岐命令，サブルーチン・コール命令をアセンブラで自動的に選択することが可能です。CA850 では，この疑似命令に相当するものはありません。

【RA78K4】

BR　　：オペランドに指定された式の値に応じて，アセンブラが自動的に 2 バイトから 4 バイトの BR 分岐命令を選択し，該当するオブジェクト・コードを出力します。
記述形式は次のとおりです。

BR　式

CALL　　：オペランドに指定された式の値に応じて，アセンブラが自動的に 3 バイトか 4 バイトの CALL 命令を選択し，該当するオブジェクト・コードを出力します。
記述形式は次のとおりです。

CALL　式

＜記述例＞

```
BR              JLABEL0
CALL    JLABEL1
```

【CA850】

ありません。

分岐命令は jr 命令，または，jarl 命令で分岐を記述してください。ただし，ディスプレースメントを越える分岐の場合には，jmp 命令を使用してください。

3. 7　汎用レジスタ選択疑似命令

この節では，汎用レジスタ選択疑似命令について説明をします。

3.7.1　RSS

CA850 では，RSS 疑似命令に相当するものはありません。

【RA78K4】

RSS　　：オペランドで指定されたレジスタ・セット選択フラグの値を元にして，プログラム中に記述された機能名称の汎用レジスタを対応する絶対名称の汎用レジスタに置き換えてオブジェクト・コードを生成します。記述形式は次のとおりです。

RSS　　0 または 1

＜記述例＞

```
RSS             0
```

【CA850】

ありません。

デバイスにこの機能はありません。

3. 8　マクロ疑似命令（マクロ疑似命令，スキップ疑似命令，繰り返しアセンブル疑似命令）

この節では，マクロ疑似命令について説明をします。

3. 8. 1　MACRO, .macro

マクロを定義する場合に，RA78K4 **では** MACRO **疑似命令を，**CA850 **では**.macro **疑似命令を使用します。**

【RA78K4】

MACRO　**：**MACRO **疑似命令と** ENDM **疑似命令の間に記述された一連の文に対し，マクロ名を付けマクロの定義を行います。**
記述形式は次のとおりです。

マクロ名　MACRO　**［仮パラメータ［［,…］］**

＜記述例＞

```
ADDMAC  MACRO             PARA1,PARA2
        MOV       A,#PARA1
        ADD       A,#PARA2
        ENDM
```

【CA850】

.macro　**：**.macro **疑似命令と**.endm **疑似命令で囲まれた文の並びを，第一オペランドで指定したマクロ名に対するマクロ本体として定義します。**
記述形式は次のとおりです。

.macro　**マクロ名**　**［仮パラメータ**,**］**...

＜記述例＞

```
        .macro   ADDMAC   PARA1,PARA2
        mov      PARA1,r10
        add      PARA2,r10
        .endm
```

3. 8. 2　LOCAL, .local

マクロ内での，ローカルなシンボルを宣言する場合に，RA78K4 **では** LOCAL **疑似命令を，**CA850 **では**.local **疑似命令を使用します。**

【RA78K4】

LOCAL　：オペランド欄に指定されたシンボル名はそのマクロ・ボディ内でのみ有効なローカル・シンボルであることを宣言します。
記述形式は次のとおりです。

LOCAL　シンボル名［,…］

＜記述例＞

```
JMAC    MACRO
        LOCAL   LAB
LAB:
        BR      $LAB
        ENDM
```

【CA850】

.local　：指定した文字列を特有の識別子として置き換えられるローカル・シンボルとして宣言します。
記述形式は次のとおりです。

.local　ローカル・シンボル［,ローカル・シンボル］...

＜記述例＞

```
        .macro  JMAC
        .local  LAB
LAB:
        jr      LAB
        .endm
```

3. 8. 3　REPT, .repeat

マクロを繰り返し展開する場合に，RA78K4 では REPT 疑似命令を，CA850 では.repeat 疑似命令を使用します。

【RA78K4】

REPT　：REPT 疑似命令と ENDM 疑似命令の間に記述された一連の文を，オペランドで指定した式の値の分だけアセンブラが繰り返し展開します。
記述形式は次のとおりになります。

REPT　絶対式

＜記述例＞

```
        REPT    3
        NOP
        ENDM
```

【CA850】

.repeat　：.repeat 疑似命令と.endm 疑似命令で囲まれた文の並びを第一オペランドで指定した絶対値式で与えられた回数だけ繰り返してアセンブルします。
記述形式は次のとおりです。

.repeat　絶対値式

＜記述例＞

```
        .repeat  3
        nop
        .endm
```

3. 8. 4　IRP, .irepeat

マクロを仮パラメータ付きで繰り返し展開する場合に，RA78K4 では IRP 疑似命令を，CA850 では.irepeat 疑似命令を使用します。

【RA78K4】

IRP　：IRP 疑似命令と ENDM 疑似命令の間にある一連の文をオペランドで指定された実パラメータで仮パラメータを置き換えながら実パラメータの数だけ繰り返し展開します。
記述形式は次のとおりです。

IRP　仮パラメータ,<実パラメータ［,…］>

＜記述例＞

```
        IRP     PARA,<0AH,0BH,0CH>
        ADD     A,#PARA
        MOV     [HL],A
        INCW    HL
        ENDM
```

【CA850】

.irepeat　：.irepeat 疑似命令と.endm 疑似命令で囲まれる文の並びをそのブロック内に現れた第一オペランドで指定した仮パラメータを第二オペランド以降に指定した実パラメータに置き換え繰り返しアセンブルします。
記述形式は次のとおりです。

.irepeat　仮パラメータ　実パラメータ［,実パラメータ］...

＜記述例＞

```
        .irepeat PARA     0xa,0xb,0xc
        add      PARA,r10
        st.b     r10,[r11]
        add      1,r11
        .endm
```

3. 8. 5　EXITM, .exitm, .exitma

RA78K4 では，マクロ・ボディの展開や繰り返しマクロの展開を強制的に終了させるために EXITM 疑似命令を使用します。CA850 では，マクロ・ボディの展開を強制的に終了させる疑似命令はありませが，繰り返しマクロの展開強制的に終了させるためには.exitm 疑似命令または.exitma 疑似命令を使用します。

【RA78K4】

EXITM　：マクロ・ボディの展開，REPT 疑似命令，IRP 疑似命令のネスティング・レベルを，そのマクロ・ボディの展開，REPT 疑似命令，IRP 疑似命令に入ったときのネスティング・レベルまで強制的に戻します。
記述形式は次のとおりです。

EXITM

＜記述例＞

```
        REPT     10
        $_IF(SW1 < 5)
                 DB       0FH
                 EXITM
        $ELSE
                 DB       00H
        $ENDIF
        $_IF(SW1 > 10)
                 DB       0AH
        $ELSE
                 DB       05H
        $ENDIF
        ENDM
```

【CA850】

.exitm　：本疑似命令を囲んでいるもっとも内側の繰り返しアセンブル疑似命令の繰り返しアセンブルをスキップします。
記述形式は次のとおりです。

.exitm

.exitma　：本疑似命令を囲んでいるもっとも外側の繰り返しアセンブル疑似命令の繰り返しアセンブルをスキップします。
記述形式は次のとおりです。

.exitma

＜記述例＞

```
        .sdata
        .repeat 10
        .if     SW1 < 5
                .byte   0xf
                .exitm
        .else
                .byte   0x0
        .endif
        .if     SW1 > 10
                .byte   0xa
        .else
                .byte   0x05
        .endif
        .endm

        .repeat 5
        .if     SW2 > 5
                .byte   0xf
                .if     SW2 < 10
                        .byte   0xa
                        .exitma
                .else
                        .byte   0x05
                .endif
        .else
                .byte   0x0
        .endif
        .endm
```

(

3. 8. 6　ENDM, .endm

マクロの終了を定義するために，RA78K4 **では** ENDM **疑似命令を，**CA850 **では**.endm **疑似命令を使用します。**

【RA78K4】

ENDM　**：マクロ機能として定義される一連のステートメントの終了をアセンブラに指示します。**
記述形式は次のとおりです。

ENDM

＜記述例＞

```
ADDMAC MACRO            PARA1,PARA2
       MOV      A,#PARA1
       ADD      A,#PARA2
       ENDM
```

【CA850】

.endm　**：繰り返し区間の終わり，またはマクロ本体の終わりを示します。**
記述形式は次のとおりです。

.endm

＜記述例＞

```
       .macro   ADDMAC   PARA1,PARA2
       mov      PARA1,r10
       add      PARA2,r10
       .endm
```

3. 9　アセンブル終了疑似命令

この節では，アセンブル終了疑似命令について説明をします。

3. 9. 1　END

RA78K4 **では，アセンブラ・ソース・モジュールの終了を** END **疑似命令で宣言します。**

CA850 **では，アセンブラ・ソース・モジュールの終了を疑似命令で宣言する必要がないので，疑似命令に相当するものはありません。**

【RA78K4】

END　**：ソース・モジュールの終了をアセンブラに宣言します。**
記述形式は次のとおりです。

END

【CA850】

ありません。

ソース・ファイルの最後に終了を意味するキーワードを記述する必要はありません。

3. 10　アセンブラ対象品種指定制御命令（アセンブラ制御疑似命令）

この節では，アセンブラ対象品種指定制御命令（アセンブラ制御疑似命令）について説明をします。

3. 10. 1　$PROCESSOR, .option

アセンブラの対象となる品種をアセンブラ・ソース・ファイル中に記述する場合に，RA78K4 では $PROCESSOR 制御命令を，CA850 では.option 疑似命令を使用します。ただし，.option 疑似命令は対象品種を指定するだけではなく，その他の機能もありますので，詳細は **CA850 アセンブリ言語編（U15027J）**を参照してください。

【RA78K4】

$PROCESSOR　　：アセンブル対象品種を指定します。

記述形式は次のとおりです。

$PROCESSOR　(**品種名**)

$PC　(**品種名**)

＜記述例＞

```
$PROCESSOR  (4038)
```

【CA850】

.option　：オペランドに指定したオプションにしたがってアセンブラを制御します。

cpu を指定し，アセンブル対象品種を指定します。

記述形式は次のとおりです。

.option　cpu　**品種名**

＜記述例＞

```
.option cpu 3003
```

3. 11　ディバグ情報出力制御命令

この節では，ディバグ情報出力制御命令について説明をします。

3. 11. 1　$DEBUG, $NODEBUG, $DEBUGA, $NODEBUGA

RA78K4 では，ディバグ情報の制御をアセンブラ・ソース・ファイル上に制御命令で指定することが可能です。

CA850 ではこの制御命令に対応する疑似命令はありませんので，ディバグ情報はオプションで指定してください。

【RA78K4】

$DEBUG	：ローカル・シンボル情報を出力します。
$NODEBUG	：ローカル・シンボル情報を出力しません。
$DEBUGA	：アセンブラ・ソース・ディバグ情報を出力します。
$NODEBUGA	：アセンブラ・ソース・ディバグ情報を出力しません。

【CA850】

ありません。

ディバグ情報を出力する際には，オプションで指定してください。

3. 12　クロス・レファレンス・リスト出力指定制御命令

この節では，クロス・レファレンス・リスト出力指定制御命令について説明をします。

3. 12. 1　$XREF, $NOXREF, $SYMLIST, $NOSYMLIST

RA78K4 では，クロス・レファレンス・リストの出力を制御する制御命令，シンボル・リストの出力を制御する制御命令があります。

CA850 ではクロス・レファレンス・リストは出力する事ができませんので，cxref を使用してクロス・レファレンス情報を参照してください。また，シンボル・リストは出力できませんので，dump850 または rammap を使用してシンボル情報を参照してください。

【RA78K4】

$XREF	：クロス・レファレンス・リストを出力します。
$NOXREF	：クロス・レファレンス・リストを出力しません。
$SYMLIST	：シンボル・リストを出力します。
$NOSYMLIST	：シンボル・リストを出力しません。

【CA850】

ありません。

クロス・レファレンス・リストはクロス・レファレンス・ツール（cxref）を使用して参照してください。

シンボルについては，ダンプ・ツール（dump850）またはメモリ視覚化ツール（rammap）を使用して参照してください。

3. 13　インクルード制御命令（ファイル入力制御疑似命令）

この節では，インクルード制御命令（ファイル入力制御疑似命令）について説明をします。

3. 13. 1　$INCLUDE, .include

インクルード・ファイルを指定する場合，RA78K4 では$INCLUDE 制御命令を，CA850 では.include 疑似命令を使用します。

【RA78K4】

$INCLUDE　　：指定されたファイルの内容を挿入展開し，アセンブルします。
記述形式は次のとおりです。

$INCLUDE（ファイル名）
$IC（ファイル名）

＜記述例＞

```
$INCLUDE(MAIN.H)
```

【CA850】

.include　：オペランドに指定したファイルの内容を，本疑似命令の置かれている位置に置かれているものとして扱います。
記述形式は次のとおりです。

.include　“ファイル名”

＜記述例＞

```
.include        "main.h"
```

3. 13. 2　.binclude

CA850 では，バイナリのファイルをインクルードすることが可能です。

【RA78K4】

ありません。

【CA850】

.binclude **：オペランドに指定したファイルの内容を，本疑似命令の置かれている位置にバイナリ・データとしてそのまま配置します。**

記述形式は次のとおりです。

.binclude　“**ファイル名**”

＜記述例＞

```
.binclude       "sub.o"
```

3. 14　アセンブル・リスト制御命令

この節では，アセンブル・リスト制御命令について説明をします。

3.14.1　$EJECT, $TITLE, $SUBTITLE, $LIST, $NOLIST, $GEN, $NOGEN, $COND, $NOCOND, $FORMFEED, $NOFORMFEED, $WIDTH, $LENGTH, $TAB

RA78K4 **では，アセンブル・リストの** 1 **行の文字数や** 1 **ページの行数などを制御する制御命令があります。**

CA850 **には，アセンブル・リストを制御する制御命令はありません。**

【RA78K4】

$EJECT	**：アセンブル・リストの改ページをアセンブラに指示します。**
$TITLE	**：アセンブル・リストの各ページのヘッダのタイトル欄に印字する文字列を指定します。**
$SUBTITLE	**：アセンブル・リストの各ページ・ヘッダのサブタイトル部に印字する文字列を指定します。**
$LIST	**：アセンブル・リストの出力開始位置をアセンブラに指示します。**
$NOLIST	**：アセンブル・リストの出力中止位置をアセンブラに指示します。**
$GEN	**：マクロ展開部をアセンブル・リストに出力するように指示します。**
$NOGEN	**：マクロ展開部をアセンブル・リストに出力しないように指示します。**
$COND	**：条件アセンブルの条件不成立部分をアセンブル・リストに出力するように指示します。**
$NOCOND	**：条件アセンブルの条件不成立部分をアセンブル・リストに出力しないように指示します。**
$FORMFEED	**：リスト・ファイルの最後にフォーム・フィードを出力することを指示します。**
$NOFORMFEED	**：リスト・ファイルの最後にフォーム・フィードを出力しないことを指示します。**

$WIDTH　　：リスト・ファイルの１行の最大文字数を指示します。
$LENGTH　　：リスト・ファイルの 1 ページの行数を指示します。
$TAB　　：リスト・ファイルのタブの展開文字数を指示します。

【CA850】

ありません。
アセンブル・リスト・ファイルを操作することはできません。

3. 15　条件付きアセンブル制御命令（条件アセンブル疑似命令）

この節では，条件付きアセンブル制御命令（条件アセンブル疑似命令）について説明をします。

3. 15. 1　$SET, $RESET

RA78K4 では，条件付きアセンブルのスイッチ名に値を定義する$SET 制御命令，$RESET 疑似命令があります。

CA850 では制御命令はありませんので，.set 疑似命令を使用してスイッチ名を定義してください。

【RA78K4】

$SET　：IF/ELSEIF 制御命令で指定するスイッチ名に真の値（0FFH）を与えます。
記述形式は次のとおりです。

$SET （スイッチ名［:スイッチ名…］）

$RESET ：IF/ELSEIF 制御命令で指定するスイッチ名に偽の値（0H）を与えます。
記述形式は次のとおりです。

$RESET （スイッチ名［:スイッチ名…］）

＜記述例＞

```
$SET    (SWSYM0)
$RESET  (SWSYM1)
```

【CA850】

ありません。
.set 疑似命令を使用してください。

＜記述例＞

```
        .set    SWSYM0,0xff
        .set    SWSYM1,0
```

3. 15. 2　$IF, .ifdef

RA78K4 では，$IF 制御命令で指定したスイッチ名が真の場合にアセンブルします。

CA850 では，.ifdef 疑似命令で指定したスイッチ名が定義されていればアセンブルします。

【RA78K4】

$IF　　：指定したスイッチ名が真（≠0）の場合，次の条件付きアセンブル疑似命令が現れるまでアセンブルされます。偽（=0）の場合は，次の条件付きアセンブル疑似命令が現れるまでアセンブルされません。

記述形式は次のとおりです。

$IF(スイッチ名［:スイッチ名…］)

＜記述例＞

```
$IF(SWSYM0)
```

【CA850】

.ifdef　　：オペランドで指定した名前が定義されている場合は，対応する.else 疑似命令，.elseif 疑似命令，.elseifn 疑似命令または.endif 疑似命令までのブロックをアセンブルします。定義されていない場合は，対応する.else 疑似命令，.elseif 疑似命令，.elseifn 疑似命令または.endif 疑似命令までのブロックをアセンブルしません。

記述形式は次のとおりです。

.ifdef　名前

＜記述例＞

```
        .ifdef   SWSYM0
```

3. 15. 3　.ifndef

CA850 では，.ifndef 疑似命令で指定したスイッチ名が定義されていなければアセンブルします。

【RA78K4】

ありません。

【CA850】

.ifndef　：オペランドで指定した名前が定義されていない場合は，対応する.else 疑似命令，.elseif 疑似命令，.elseifn 疑似命令または.endif 疑似命令までのブロックをアセンブルします。定義されている場合は，対応する.else 疑似命令，.elseif 疑似命令，.elseifn 疑似命令または.endif 疑似命令までのブロックをアセンブルしません。
記述形式は次のとおりです。

.ifndef　名前

＜記述例＞

```
        .ifndef  SWSYM3
```

3. 15. 4　$_IF, .if

RA78K4 では，$_IF 制御命令の条件式がが真の場合にアセンブルします。
CA850 では，.if 疑似命令の式が真の場合にアセンブルします。

【RA78K4】

$_IF　：指定した条件式が真（≠0）の場合，次の条件付きアセンブル疑似命令が現れるまでアセンブルされます。偽（=0）の場合は，次の条件付きアセンブル疑似命令が現れるまでアセンブルされません。
記述形式は次のとおりです。

$_IF　条件式

＜記述例＞

```
    $_IF SWSYM2 = 0
```

【CA850】

.if　：オペランドで指定した絶対値式が真（≠0）の場合は，対応する.else 疑似命令，.elseif 疑似命令，.elseifn 疑似命令または.endif 疑似命令までのブロックをアセンブルします。偽（=0）の場合は，対応する.else 疑似命令，.elseif 疑似命令，.elseifn 疑似命令または.endif 疑似命令までのブロックをアセンブルしません。
記述形式は次のとおりです。

.if　絶対値式

＜記述例＞

```
        .if      SWSYM2==0
```

3. 15. 5　.ifn

CA850 では，.ifndef 疑似命令の式が偽の場合にアセンブルします。

【RA78K4】

ありません。

【CA850】

.ifn　：オペランドで指定した絶対値式が偽（=0）の場合は，対応する.else 疑似命令，.elseif 疑似命令，.elseifn 疑似命令または.endif 疑似命令までのブロックをアセンブルします。真（≠0）の場合は，対応する.else 疑似命令，.elseif 疑似命令，.elseifn 疑似命令または.endif 疑似命令までのブロックをアセンブルしません。
記述形式は次のとおりです。

.ifn　絶対値式

＜記述例＞

```
        .ifn     SWSYM2==5
```

3. 15. 6　$ELSEIF, $_ELSEIF, .elseif

それ以前の条件付きアセンブル制御命令の条件が不成立の場合に条件判断され，真のときにアセンブルする場合に，RA78K4 ではシンボルの場合には$ELSEIF 制御命令を，式の場合には$_ELSEIF 制御命令を使用しますが，CA850 では，.elseif 疑似命令を使用します。

【RA78K4】

$ELSEIF　：それ以前に記述してあるすべての条件付きアセンブル制御命令の条件が不成立の場合のみ条件判定が行われます。
指定したスイッチ名が真（≠0）の場合，次の条件付きアセンブル疑似命令が現れるまでアセンブルされます。偽（=0）の場合は，次の条件付きアセンブル疑似命令が現れるまでアセンブルされません。
記述形式は次のとおりです。

$ELSEIF(スイッチ名［:スイッチ名…］)

$_ELSEIF　：それ以前に記述してあるすべての条件付きアセンブル制御命令の条件が不成立の場合のみ条件判定が行われます。
指定した条件式が真（≠0）の場合，次の条件付きアセンブル疑似命令が現れるまでアセンブルされます。偽（=0）の場合は，次の条件付きアセンブル疑似命令が現れるまでアセンブルされません。
記述形式は次のとおりです。

$_ELSEIF　**条件式**

＜記述例＞

```
$ELSEIF(SWSYM1)
$_ELSEIF        SWSYM2 = 0FFH
```

【CA850】

.elseif　**：それ以前に記述してあるすべての条件アセンブル疑似命令の条件が不成立の場合のみ条件判定が行われます。**
オペランドで指定した絶対値式が真（≠0）の場合は，対応する.else **疑似命令，**.elseif **疑似命令，**.elseifn **疑似命令または**.endif **疑似命令までのブロックをアセンブルします。偽（**=0**）の場合は，対応する**.else **疑似命令，**.elseif **疑似命令，**.elseifn **疑似命令または**.endif **疑似命令までのブロックをアセンブルしません。**
記述形式は次のとおりです。

.elseif　**絶対値式**

＜記述例＞

```
        .elseif  SWSYM2==0xff
```

3. 15. 7　.elseifn

CA850 **では，**.elseifn **疑似命令を使用することにより，それ以前の条件付きアセンブル制御命令の条件が不成立の場合に条件判断され，オペランドに指定された式が偽の場合にアセンブルされます。**

【RA78K4】

ありません。

【CA850】

.elseifn　**：それ以前に記述してあるすべての条件アセンブル疑似命令の条件が不成立の場合のみ条件判定が行われます。オペランドで指定した絶対値式が偽（**=0**）の場合は，対応する**.else **疑似命令，**.elseif **疑似命令，**.elseifn **疑似命令または**.endif **疑似命令までのブロックをアセンブルします。真（≠0）の場合は，対応する**.else **疑似命令，**.elseif **疑似命令，**.elseifn **疑似命令または**.endif **疑似命令までのブロックをアセンブルしません。**
記述形式は次のとおりです。

.elseifn　**絶対値式**

＜記述例＞

```
        .elseifn SWSYM2==0xff
```

3. 15. 8　$ELSE, .else

それ以前のすべての条件付きアセンブルが偽の場合に，RA78K4 では$ELSE 制御命令を，CA850 では.else 疑似命令を使用することにより，アセンブルすることを指定できます。

【RA78K4】

$ELSE　：それ以前に記述してあるすべての条件付きアセンブル制御命令の条件が不成立の場合，ELSE 制御命令以降 ENDIF 制御命令が現れるまでアセンブルします。
記述形式は次のとおりです。

```
$ELSE
```

【CA850】

.else　：それ以前に記述してあるすべての条件アセンブル疑似命令の条件が不成立の場合，.else 疑似命令と本疑似命令に対応する.endif とで囲まれる文の並びをアセンブルします。
記述形式は次のとおりです。

```
.else
```

3. 15. 9　$ENDIF, .endif

条件付きアセンブルの終了を定義するには，RA78K4 では$ENDIF 制御命令を，CA850 では.endif 疑似命令を使用します。

【RA78K4】

$ENDIF　：条件付きアセンブルの対象となるソース・ステートメントの終了をアセンブラに指示します。
記述形式は次のとおりです。

```
$ENDIF
```

【CA850】

.endif　：条件アセンブル疑似命令による制御の範囲の終わりを示します。
記述形式は次のとおりです。

```
.endif
```

3. 16　SFR 領域変更制御命令

この節では，SFR 領域変更制御命令について説明をします。

3. 16. 1　$CHGSFR, $CHGSFRA

78K/**IV**シリーズでは，SFR 領域のアドレスを LOCATION 命令によって変更することが可能です。その際に，SFR 領域変更制御命令で指定します。

V850 ファミリには，このような機能がないので，SFR 領域変更制御命令に相当する疑似命令はありません。

【RA78K4】

$CHGSFR　　：オペランドの絶対式により SFR 領域のアドレスを指定します。
記述形式は次のとおりです。

$CHGSFR　(絶対式)

$CHGSFRA　　：SFR 領域がどちらの場合にもリンクできるオブジェクトを作成するように指示します。
記述形式は次のとおりです。

$CHGSFRA

【CA850】

ありません。

3. 17　漢字コード指定制御命令

この節では，漢字コード指定制御命令について説明をします。

3. 17. 1　$KANJICODE

RA78K4 では，コメントに記述した漢字のコードを指定することが可能です。

CA850 ではこの制御命令にに相当する疑似命令はありませんが，-Xk オプションで指定することが可能です。

【RA78K4】

$KANJICODE　　：コメントに記述された漢字を，指定された漢字コードとして解釈します。
記述形式は次のとおりです。

$KANJICODE　漢字コード

【CA850】

ありません。
漢字コードを指定することはできません。

第4章　リンク・ディレクティブ編

この章では，リンカ（リンク・エディタ）でのリンク・ディレクティブについて，注意する点および基本的な記述方法について説明します。

4.1　リンク・ディレクティブの内容

RA78K4 の lk78k4 と CA850 の ld850 のリンク・ディレクティブ・ファイルに記述する内容は次の表のようになります。

表4－1　リンク・ディレクティブ

RA78K4	CA850
メモリ・ディレクティブ セグメント配置ディレクティブ	セグメント・ディレクティブ マッピング・ディレクティブ シンボル・ディレクティブ

RA78K4 ではディレクティブをアドレス順に記述する必要はありませんが，CA850 ではアドレス順に記述する必要がありますので，注意してください。

また，RA78K4 ではシンボルを生成するためのシンボル・ディレクティブはありませんが，CA850 では TP シンボル，GP シンボルおよび EP シンボルを生成するために，シンボル・ディレクティブの記述が必要です。

4. 2　リンク・ディレクティブの記述

RA78K4，CA850 のリンク・ディレクティブの記述形式は次のようになります。

【RA78K4】

メモリ・ディレクティブ

MEMORY **メモリ領域名**:(**スタート・アドレス値**,**サイズ**)［/**メモリ空間名**］

セグメント配置ディレクティブ

MERGE **セグメント名**:［AT (**スタート・アドレス**)］

［=**メモリ領域名指定**］［/**メモリ空間名**］

＜記述例＞

```
memory TBL        : ( 000000h , 00080h )
memory ROM        : ( 000080h , 0BF80h )
memory RAM1       : ( 0FF700h , 00600h )
memory STK        : ( 0FFD00H, 00020h )
memory SDR        : ( 0FFD20h , 000E0h )
memory SDR1       : ( 0FFE00h , 00080h )
memory RAM        : ( 0FFE80h , 00180h )

merge @@INIT      : = RAM1
merge @@DATA      : = RAM1
merge @@INIS      : = SDR
merge @@DATS      : = SDR
merge @@BITS      : = SDR
merge @@INIS1     : = SDR1
merge @@DATS1     : = SDR1
merge @@BITS1     : = SDR1

memory EXMEM : (050000H, 1000H)

merge  DAT1 := EXMEM
```

【CA850】

セグメント・ディレクティブ

セグメント名:!**セグメントタイプ** ?**セグメント属性**［V **アドレス**］

［L **最大メモリ・サイズ**］［H **ホール・サイズ**］［F **フィリング値**］

［A **整列条件**］｛**マッピング・ディレクティブ**｝;

マッピング・ディレクティブ

セクション名=$**セクション・タイプ** ?**セクション属性**［**セクション名**］

［V **アドレス**］［H **ホール・サイズ**］［A **整列条件**］

［｛**ファイル名**［**ファイル名**］...｝］;

シンボル・ディレクティブ

シンボル名@%**シンボル種別**［&**ベース・シンボル名**］［V **アドレス**］

［A **整列条件**］［｛**セグメント名**［**セグメント名**］...｝］;

＜記述例＞

```
SCONST     : !LOAD ?R {
  .sconst            = $PROGBITS       ?A .sconst;
};

TEXT       : !LOAD ?RX {
  .pro_epi_runtime = $PROGBITS         ?AX;
  .text               = $PROGBITS      ?AX;
};

DATA       : !LOAD ?RW V0x100000 {
  .data              = $PROGBITS       ?AW.data;
  .sdata             = $PROGBITS       ?AWG.sdata;
  .sbss              = $NOBITS         ?AWG.sbss;
  .bss               = $NOBITS         ?AW.bss;
};

CONST      : !LOAD ?R {
  .const             = $PROGBITS       ?A .const;
};

SEDATA     : !LOAD ?RW V0xff6000 {
  .sedata            = $PROGBITS       ?AW .sedata;
  .sebss             = $NOBITS         ?AW .sebss;
};

SIDATA     : !LOAD ?RW V0xffe000 {
  .tidata.byte       = $PROGBITS       ?AW .tidata.byte;
  .tibss.byte        = $NOBITS         ?AW .tibss.byte;
  .tidata.word       = $PROGBITS       ?AW .tidata.word;
  .tibss.word        = $NOBITS         ?AW .tibss.word;
  .tidata            = $PROGBITS       ?AW .tidata;
  .tibss             = $NOBITS         ?AW .tibss;
  .sidata            = $PROGBITS       ?AW .sidata;
  .sibss             = $NOBITS         ?AW .sibss;
};

__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;
```

第5章　翻訳限界

この章では，CC78K4 と CA850 のコンパイル時の翻訳限界値について比較します。

比較の結果は，次の表のようになります。

表5－1　翻訳限界値

項番	項　　目	CC78K4	CA850
1	複文，繰り返し制御文，選択制御文のネスト	45	127
2	条件コンパイルのネスト	255	255
3	1 つの宣言中の 1 つの算術型，構造体型，共用体型または不完全型を修飾する（任意の組み合わせ）ポインタ，配列および関数宣言子の個数	12	16
4	完全な宣言子の中の，かっこで囲まれた宣言子の入れ子のレベル数	591	255
5	完全な式の中の，かっこで囲まれた式の入れ子のレベル数	32	255
6	マクロ名における有効先頭文字数	31	1023
7	外部シンボル名における有効先頭文字数	30	1022
8	内部シンボル名における有効先頭文字数	30	1023
9	1 ソース・モジュール・ファイル中のシンボル数	1024	4095
10	1 ブロックでブロック・スコープを持つシンボル数	255	
11	1 ソース・モジュール・ファイル中のマクロ数	10000	2047 注
12	1 つの関数定義，関数呼び出しのパラメータ	39	255
13	1 つのマクロ定義，マクロ呼び出しのパラメータ	31	127
14	1 つの論理ソース行内の文字数	509	32766
15	結合後の文字列リテラル内の文字数	509	32766
16	インクルード・ファイルのネスト	8	50
17	switch 文の case レーベル数	257	1025
18	1 つの構造体または共用体のメンバ数	127	1023
19	1 つの列挙の列挙定数の数	255	1023
20	1 つの構造体または共用体における構造体または共用体のネスト	15	63

注　Cコンパイラ・オプション（-Xm）で変更することができます（最大 32767）。

付　録　総合索引

50 音で始まる語句の索引

アルファベットで始まる語句の索引

〔メ　モ〕

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）

電　話　：044−435−9494
FAX　：044−435−9608
E-mail　：s-info@saed.tmg.nec.co.jp

【営業関係お問い合わせ先】

第一販売事業部
東　京　(03)3798−6106, 6107, 6108
大　阪　(06)6945−3178, 3200, 3208, 3212
仙　台　(022)267−8740
郡　山　(024)923−5591
千　葉　(043)238−8116

第二販売事業部
東　京　(03)3798−6110, 6111, 6112
立　川　(042)526−5981, 6167
松　本　(0263)35−1662
静　岡　(054)254−4794
金　沢　(076)232−7303
松　山　(089)945−4149

第三販売事業部
東　京　(03)3798−6151, 6155, 6586, 1622, 1623, 6156
水　戸　(029)226−1702
広　島　(082)242−5504
前　橋　(027)243−6060
鳥　取　(0857)27−5313
太　田　(0276)46−4014
名古屋　(052)222−2170, 2190
福　岡　(092)261−2806

【資料の請求先】

上記営業関係お問い合わせ先またはNEC特約店へお申しつけください。

【NECエレクトロンデバイス　ホームページ】

NECエレクトロンデバイ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.ic.nec.co.jp/

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］CC78K4 から CA850 への移植ガイド　アプリケーション・ノート
（U15653JJ1V0AN00（第1版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）　（　　）
ご住所　（　　）
お電話番号　（　　）
お仕事の内容　（　　）
お名前　（　　）

1．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

2．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

3．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）
理由［　　］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは
NEC 販売員，特約店販売員，　その他（　　）

ご協力ありがとうございました。
下記あてに FAX で送信いただくか，最寄りの販売員にコピーをお渡しください。

日本電気(株) NEC エレクトロンデバイス
半導体テクニカルホットライン
FAX：（044）435-9608

2000.6

キリトリ