開発環境移行ガイド

V850 から RH850 への移行（コンパイラ編）

本書では、V850 用コンパイラ（以降、CA850）から、RH850 ファミリ用コンパイラ（以降、CC-RH）へ移行する際に主に注意すべき点について説明します。

目次:

第1章　　オプション

本章では、CA850 のオプションに相当する CC-RH のオプションを比較表にて示します。

なお、CC-RH では、コンパイル・オプション、アセンブル・オプションの大文字／小文字は区別しますが、リンク・オプションの大文字と小文字は区別しません。CA850 は大文字／小文字は区別します。

1.1　コンパイル・オプション

分類	説明	CA850(ca850.exe)	CC-RH(ccrh.exe)
バージョン／ヘルプ表示／動作状態	バージョン情報を表示して終了	-V	-V
	オプション情報を表示して終了	-help	-h
	コンパイル状況の詳細を表示	-v	なし
出力ファイル指定	中間言語ファイルの保存先を指定	-Fic	なし
	オブジェクト・ファイルの保存先を指定	-Fo	-Xobj_path
	アセンブリ・ファイルの保存先を指定	-Fs	-Xasm_path
	アセンブル・リスト・ファイルの保存先を指定	-Fv	-Xasm_option=-Xprn_path
	出力ファイル名を指定	-o	-o
	作業用フォルダを指定	-temp	なし
ソース・デバッガ制御	ld.w/ld.h/st.w/st.h命令を1ビット操作命令へ置き換える動作を禁止	-Xno_word_bitop	なし（注1）
	デバッグ情報を出力	-g	-g
デバイス指定	メモリ空間を256Mバイトに変更	-X256M	なし
	プログラマブル周辺I/O レジスタのアドレスを設定	-Xbpc	なし
	V850コア共通のマジックナンバを埋め込み	-cn	なし
	V850Exコア共通のマジックナンバを埋め込み	-cnv850e	
	V850E2コア共通のマジックナンバを埋め込み	-cnv850e2	
	ターゲット・デバイスを指定	-cpu	なし（注2）
	デバイス・ファイルを検索するフォルダを指定	-Xdev_path	
コンパイラ制御指定	コンパイル処理まで実行	-S	-S
	アセンブル・リストを出力	-a	-Xasm_option=-Xprn_path
	アセンブル処理まで実行	-c	-c
	フロントエンドのみ実行	-m	なし
ROM化制御	ROM化用のオブジェクトを作成	-Xr	-Xlk_option=-rom

分類	説明	CA850(ca850.exe)	CC-RH(ccrh.exe)
プリプロセッサ制御設定	前処理の出力にソース・プログラムのコメント出力	-C	-Xprerocess=comment
	プリプロセッサ・マクロを定義	-D	-D
	前処理のみ実行し結果を標準出力に出力	-E	なし
	インクルード・ファイルを検索するフォルダを指定	-I	-I
	前処理のみ実行し結果をファイルに出力	-P	-P
	プリプロセッサ・マクロの定義を解除	-U	-U
	アセンブラ・マクロの定義	-Wa,-D	-D
	アセンブラ・ソース・ファイルのヘッダ・ファイルの検索フォルダを指定	-Wa,-I,	-I
	「//」によるコメントの有効化	-Xcxxcom	常に有効
	自動変数ではない変数アドレスまたは関数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	-Xd	なし
	マクロ識別子数の上限を指定	-Xm	なし（注3）
	トライグラフ系列の置換	-t	常に有効
コンパイル時のメモリ節約	コンパイル時のプリオプティマイザのメモリ使用量節約	-Wp,-D	なし
	コンパイル時の機種依存最適化のメモリ使用量節約	-Wi,-D	
エラー出力指定	エラー・メッセージをファイルに追加保存	+err_file	なし
	エラー・メッセージをファイルに上書き保存	-err_file	-Xerror_file
	エラー・メッセージの最大出力数を指定	-err_limit	なし（注4）
拡張機能指定	78K用CコンパイラCC78Kx 互換の拡張機能を有効	-cc78k	なし（注5）
最適化	デバッグ優先オプション	-Od	-Onothing
	デフォルト最適化オプション	-Ob	-Odefault
	標準最適化オプション	-Og	なし
	高度な最適化オプション	-O	なし
	より高度な最適化（オブジェクト・サイズ優先）オプション	-Os	-Osize
	より高度な最適化（実行速度優先）オプション	-Ot	-Ospeed
ターゲット・コード最適化	最も強い最適化を実施	-Wi,-O4	なし
	分岐先ラベルを整列する最適化を抑止	-Wi,-P	デフォルトで抑止（注6）

分類	説明	CA850(ca850.exe)	CC-RH(ccrh.exe)
ファイル・マージ	複数ファイル同時指定時にファイルをマージ	-Om	-Xmerge_files
インライン展開最適化制御	スタック・サイズによるインライン展開の制御	-Wp,-G	なし
	中間言語サイズによるインライン展開の制御	-Wp,-N	-Oinline_size（注7）
	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	-Wp,-S	なし
	関数の情報を標準出力に出力、またはファイルに追加出力	-Wp,-I	なし
	#pragma inline指定した関数のみインライン展開	-Wp,-inline	-Oinline=1
	#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	-Wp,-no_inline	-Oinline=0
	未参照関数の削除	-Wp,-r	なし
ループ展開最適化制御	for，while などでループを指定回展開	-Wo,-Ol	-Ounroll
	ループ展開数を指定した回数で固定してループ展開	-Wo,-Xlo	なし
strcpy，strcmp 展開	配列および構造体の整列条件を4 バイトとし、strcpy()，またはstrcmp() の呼び出しをインライン展開	-Xi	-Xinline_strcpy（注8）
外部変数ソート	外部変数をアライメントの大きい順に並び替え	-Wo,-Op	なし
分岐命令制御	分岐命令をコード・サイズ優先で並べてコードを出力	-Wo,-XFo	なし
レジスタ使用制御	外部変数を指定レジスタに割り付け	-r	なし
	コンパイラが使用するレジスタを制限	-reg	-Xreg_mode（注9）
	マスク・レジスタ機能を使用	-Xmask_reg	なし（注10）
プロローグ／エピローグ処理制御	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	-Xpro_epi_runtime	なし（注11）
型制御	型指定子の付かない単なる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	-Xbitfield	なし
	char型に対して符号を指定	-Xchar	なし（注12）
	列挙型に対しどの整数型と整合するかを指定	-Xenum_type	-Xenum_type（注13）

分類	説明	CA850(ca850.exe)	CC-RH(ccrh.exe)
変数配置制御	指定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	-G	-Xsection（注14）
	const属性のデータ、文字列リテラルを.sconstセクションに配置	-Xsconst	
	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	-Xcre_sec_data	-Omap -Osmap（注15）
		-Xcre_sec_data_only	
	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・ファイル名を指定	-Xsec_file	
switch-case 文 出力コード制御	switch 文のコード出力方式を指定	-Xcase	-Xswitch
	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	-Xword_switch	なし（注16）
構造体パッキング制御	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	-Xbyte	なし
	構造体パッキング	-Xpack	-Xpack
far jump出力制御	指定された関数への分岐に対してjmp命令を使用	-Xfar_jump	-Xfar_jump
	割り込みベクタテーブルに対してjmp命令を生成	-Xj	なし（注17）
コメント出力	出力するアセンブラ・ソース・ファイル中にC ソース・プログラムをコメントとして出力	-Xc	-Xpass_source
ANSI規約	mulh,divh命令の使用の抑止	-Xe	なし
	変数の仮定義を定義とみなす	-Xdefvar	なし
	ANSI 規格に厳密にあわせて処理	-ansi	-Xansi
日本語文字列制御	入力ファイル中の日本語のコメント、文字列に対して使用する文字コードを指定	-Xk	-Xcharacter_set
	日本語文字列を指定したコードに変換して出力	-Xkt	なし

分類	説明	CA850(ca850.exe)	CC-RH(ccrh.exe)
ライブラリ指定	ライブラリの検索フォルダを指定	-L	なし（注18）
	使用するスタート・アップ・モジュールを指定	-R	なし
	リンカで参照するアーカイブ・ファイルを指定	-l	-Xlk_option=-library=*file*
警告メッセージ制御	警告メッセージのレベル、出力、抑制を指定	-w	なし
	指定した番号の警告メッセージを出力	-won	なし
	指定した番号の警告メッセージを抑制	-woff	-Xno_warning
コマンド・ファイル指定	コマンド・ファイルを指定	@	@
CPU 不具合パッチ	CPUの不具合に対応したコードを出力	-Xv850patch	-Xpatch
各モジュール	モジュールに対するオプションを指定	-W	-Xasm_option -Xlk_option（注19）
その他	強力な最適化	+Oc	なし

注1：ビット操作命令の生成規則につきましては、コンパイラ編マニュアル（R20UT3516JJ0102）の「11.3 ビット操作命令の出力を制御する方法 【V1.05.00 以降】」を参考にしてください。
注2：デバイス・ファイルはサポートしていません。
注3：動作するホスト・マシンのメモリに依存します。
注4：上限はありません。
注5：SHC用にコーディングしたCソース・ファイルをチェックするオプション”-Xcheck=shc”があります。
注6：-Xalign4=all オプションを指定すると整列が有効になります。
注7：コード・サイズが何%増加するまでインライン展開を行うかを指定します。
注8：memcpy, memset 関数もインライン展開の対象です。
注9：CC-RHには26レジスタ・モードはありません。
注10：RH850はハード側でゼロ拡張するためマスク・レジスタ機能はありません。
注11：関数のプロローグ・エピローグ処理に対して、ランタイム・ライブラリは常に使用しません。
注12：符号が付かない単なるchar 型に対して常に符号付きとして扱います。
注13：型の指定はできません。自動で判別します。
注14：CC-RH V1.02.00で追加したオプションです。変数のデフォルト配置セクションを一括変更します。
注15：セクション・ファイルはありません。-Omap/-Osmapオプションを指定すると、アクセス頻度の高い変数をEP相対1命令アクセスに最適化します。
注16：分岐テーブルの幅を自動で判別します。
注17：ベクタテーブルはユーザーが定義する必要があります。
注18：-library オプションで指定したものを優先してリンクします。その後、未解決のシンボルがある場合、環境変数 HLNK_LIBRARY1, HLNK_LIBRARY2, HLNK_LIBRARY3 の順にライブラリを検索します。
注19：-Xasm_optionは引数をアセンブラに、-Xlk_optionは引数をリンカに渡します。

1.2　アセンブル･オプション

分類	説明	CA850(as850.exe)	CC-RH(asrh.exe)
ファイル	アセンブル・リストを生成	-a	-Xprn_path（注1）
	エラー・メッセージをファイルに追加保存	+err_file	なし
	エラー・メッセージをファイルに上書き保存	-err_file	-Xerror_file
	アセンブル・リスト・ファイル名の指定	-l	-Xprn_path
アセンブラ	定義マクロの定義	-D	-D
	外部ラベルへのアクセスに対する命令展開の制御	-G	なし（注2）
	インクルード・ファイルを検索するフォルダを指定	-I	-I
	マスク・レジスタ機能を使用	-m	なし（注3）
	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	-O	なし
	アセンブラの実行状況の詳細を標準エラー出力に出力	-v	なし
	特定の警告メッセージの出力、抑止を指定	-w	なし
	命令に22/32 を記述しない分岐命令（jarl，jr）を32ビット分岐命令とみなす	-Xfar_jump	-Xasm_far_jump

分類	説明	CA850(as850.exe)	CC-RH(asrh.exe)
デバイス	メモリ空間を256M バイトに設定	-X256M	なし
	プログラマブル周辺I/O レジスタのアドレスを設定	-bpc	なし
警告メッセージ制御	指定した番号の警告メッセージを抑制	-woff	-Xno_warning
その他	V850 コア共通のマジックナンバを埋め込み	-cn	なし
	V850Ex コア共通のマジックナンバを埋め込み	-cnv850e	
	V850E2 コア共通のマジックナンバを埋め込み	-cnv850e2	
	ターゲット・デバイスを指定	-cpu	なし（注4）
	デバイス・ファイルの置かれているフォルダを指定	-F	
	デバッグ情報を出力	-g	-g
	出力ファイル名を指定	-o	-o
	CPU の不具合を回避するためのコードを出力	-p	なし
	アセンブラのバージョン情報を標準エラー出力に出力して終了	-V	-V
	再リンク機能の使用	-zf	なし
	コマンド・ファイル指定	@	@

注1：標準出力ではなく、アセンブル・リスト・ファイルに出力します。

注2：オプションでは制御できません。ソース記述のみで判定します。

注3：RH850はハード側でゼロ拡張するためマスク・レジスタ機能はありません。

注4：デバイス・ファイルはサポートしていません。

1.3 リンク・オプション

分類	説明	CA850(ld850.exe)	CC-RH(rlink.exe)
入力ファイル	リンク・ディレクティブの指定	-D	-start（注1）
	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	-Xolddir	
出力ファイル	エラー・メッセージをファイルに追加保存	+err_file	なし
	エラー・メッセージをファイルに上書き保存	-err_file	なし（注2）
	生成するオブジェクト・ファイル名を指定	-o	-output
	リンク・マップを出力	-m	-list -show
	旧型式のリンク・マップを出力	-mo	-list -show
ライブラリ	-lオプションによって指定されたライブラリ・ファイルを検索するフォルダの指定	-L	なし（注3）
	コンパイラの標準ライブラリをリンク	-lc	-library（注4）
	コンパイラの数学ライブラリをリンク	-lm	
	入力ライブラリファイルの指定	-l	-library
再リンク機能	再リンク機能用のオブジェクト・ファイルを生成	-ext_table	なし（注5）
	ブート領域側のオブジェクト・ファイルを指定	-zf	
デバイス	メモリ空間を256M バイトに設定	-X256M	なし
	“セキュリティID”を設定	-Xsid	なし
	オプション・バイトの生成を抑止	-Xob=none	なし（注6）

分類	説明	CA850(ld850.exe)	CC-RH(rlink.exe)
リンカ	コンパイラに対して指定する[sdata／sbss のデータ配置]オプション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	-A	なし
	2パス・モードでリンク	-B	なし
	リロケーション処理のエラーメッセージを警告メッセージに変更してリンクを続行	-E	-change_message（注7）
	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	-M	なし
	リンク時の外部シンボルのサイズおよび整列条件のチェック抑止	-T	なし
	内蔵ROM･RAMをオーバーフローした際のチェックの制御を実施	-Ximem_overflow=warning	-cpu（注8）
	エントリ・ポイント・アドレスを指定	-e	-entry
	セクション間のアライン・ホールのフィリング値を指定	-f	-space
	マスク・レジスタ機能使用有無の混在チェック	-mc	なし（注9）
	異なるレジスタ・モードの混在チェック	-rc	なし（注10）
	-lオプションで指定したライブラリ･ファイルの再参照	-rescan	デフォルトで再参照
	内蔵ROM 領域への配置に対するチェック抑止	-romless	なし（注11）
	デバッグ情報等の削除	-s	-nodebug -strip
	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、整列条件のチェック抑止	-t	なし
	リンカの実行状況の詳細出力	-v	なし
	警告メッセージを抑止	-w	-change_message=information=<番号> -nomessage（注12）

分類	説明	CA850(ld850.exe)	CC-RH(rlink.exe)
その他	デバイス・ファイルを指定	-F	なし（注13）
	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	-cpu	
	バージョン情報を標準エラー出力して終了	-V	なし（注14）
	オプションの説明を標準エラー出力に出力して終了	-help	
	旧関数呼び出しと現バージョンの呼び出し仕様の混在チェック	-fc	なし
	マスク・レジスタ機能用の標準ライブラリを参照	-mask_reg	なし（注9）
	リロケータブルなオブジェクト・ファイルを生成	-r	-form=relocate
	リロケータブルなオブジェクト・ファイルを旧マッピング方式（CA850 Ver.2.30 以前）で生成	-ro	
	各レジスタ・モード用の標準ライブラリを参照	-reg	なし（注15）
	コマンド・ファイルを指定	@	-subcommand

注 1：リンク・ディレクティブ・ファイルはサポートしていません。
注 2：エラーメッセージをリンク・マップ・ファイルに表示します。
注 3：-library オプションで指定したものを優先してリンクします。その後、未解決のシンボルがある場合、環境変数 HLNK_LIBRARY1, HLNK_LIBRARY2, HLNK_LIBRARY3 の順にライブラリを検索します。
注 4：V1.02.00 以前と以降で標準ライブラリの構成が異なります。また、使用するライブラリ関数や使用するオプション条件によって引数に指定するライブラリ名も異なります。詳細はコンパイラ編マニュアルの「7.1 提供ライブラリ」をご参照ください。

注5：再リンク機能はありません。

注6：CC-RHではオプション・バイトは指定出来ません。OPTB0はCS+の[フラッシュ・オプション設定]タブ->[フラッシュ・オプション]プロパティで指定してください。
注 7：-change_message オプションによりエラーを警告に変更して処理を継続可能です。

注 8：引数にアドレスを指定します。デフォルトはエラーとしますので、対象のエラー番号を-change_messageで警告に変更してください。

注9：RH850はハード側でゼロ拡張するためマスク・レジスタ機能はありません。

注10：レジスタ・モードの異なるファイルを入力するとE0562408エラーになります。

注11：デフォルトで配置チェックは行いません。

注12：-change_messageオプションによりインフォメーション・メッセージに変更し、-nomessageオプシ

ョンによってインフォメーション・メッセージを抑止することで非表示にできます。

注13：デバイス・ファイルはサポートしていません。

注14：コマンドライン上でrlink [ENTERキー] の実行で表示されます。

注15：標準ライブラリをレジスタ・モード毎に用意していません。-Xreg_mode=commonで生成した標準ライブラリを参照します。

1.4　ヘキサコンバータ･オプション

分類	説明	CA850(hx850.exe)	CC-RH(rlink.exe)
ファイル	エラー・メッセージをファイルに追加保存	+err_file	なし
	エラー・メッセージをファイルに上書き保存	-err_file	なし（注1）
	指定ファイルにヘキサ変換した結果を出力	-o	-output
フォーマット	最大ブロック長を指定	-b	-byte_count（注2）
	出力するアドレスのオフセットを指定	-d	なし
	ヘキサ・フォーマットを指定	-f	-form
	指定されるセクションのコードを変換	-I	-output（注3）
	シンボル・テーブルを変換	-S	なし
	指定アドレスから指定されたサイズ領域のすべてのコードをヘキサ変換	-U	-output（注4） -space
	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	-x	なし
	デバイス・ファイルに定義された内蔵ROM領域の情報を使用せず変換	-rom_less	なし（注4）（注5）
	セクション・タイプNOBITSとセクション属性Aを持つセクションに対しnull文字を生成	-z	なし
その他	デバイス・ファイルの検索フォルダ	-F	なし（注5）
	バージョン情報を標準エラー出力して終了	-V	なし（注6）
	コマンド・ファイルを指定	@	-subcommand

注1：エラーメッセージをリンク・マップ・ファイルに表示します。

注2：インテル拡張ヘキサ・ファイルを指定した場合のみ有効です。

注3：-output=*file*=*section* の形式でセクション*section*を指定します。

注4：-output=*file*=*address1*-*address2* の形式でアドレス*address*を指定します。

注5：デバイス・ファイルはサポートしていません。

注6：CC-RHはリンカ（rlink.exe）がヘキサ変換します。rlinkのバージョン情報はコマンドライン上でrlink [ENTERキー] の実行で表示されます。

第2章　組み込み関数

本章では、CA850の組み込み関数に相当するCC-RHの組み込み関数を比較表にて示します。

なお、CA850にあってCC-RHにない組み込み関数を呼び出した場合、CC-RHでは通常関数としてコンパイルします。定義が無い場合はリンク時にエラーとなります。

命令	説明	CA850	CC-RH
di	割り込み禁止	void __DI(void);	void __DI(void);
ei	割り込み許可	void __EI(void);	void __EI(void);
nop	ノー・オペレーション	void __nop(void);	void __nop(void);
halt	プロセッサの停止	void __halt(void);	void __halt(void);
satadd	飽和加算	long a, b; long __satadd(a, b);	long a, b; long __satadd(a, b);
satsub	飽和減算	long a, b; long __satsub(a, b);	long a, b; long __satsub(a, b);
bsh	ハーフワード・データのバイト・スワップ	long a; long __bsh(a);	long a; long __bsh(a);
bsw	ワード・データのバイト・スワップ	long a; long __bsw(a);	long a; long __bsw(a);
hsw	ワード・データのハーフワード・スワップ	long a; long __hsw(a);	long a; long __hsw(a);
sxb	バイト・データの符号拡張	char a; long __sxb(a);	なし
sxh	ハーフワード・データの符号拡張	short a; long __sxh(a);	なし
mul	mul 命令を用いて乗算結果の上位32 ビットを変数に代入する命令	long a, b; long __mul32(a, b);	long a, b; long __mul32(a, b);
mulu	mulu 命令を用いて符号なし乗算結果の上位32ビットを変数に代入する 命令	unsigned long a, b; unsigned long __mul32u(a,b);	unsigned long a, b; unsigned long __mul32u(a, b);
sasf	論理左シフト付きフラグ条件の設定	long a; unsigned int b; long __sasf(a, b);	なし
—	割り込みレベルの制御（注1）	int NUM; void __set_il(NUM, “割り込み要求名”);	int NUM; void* ADDR; void __set_il_rh(NUM, ADDR);

注1：CA850はデバイス・ファイルで定義された文字列を“割り込み要求名”に指定します。CC-RHはADDRに割り込み制御レジスタのアドレスを指定します。

第3章　既定義マクロ

本章では、CA850の既定義マクロに相当するCC-RHの既定義マクロを比較表にて示します。

CA850 のマクロ名	CA850 の定義	CC-RH のマクロ名
__LINE__	その時点でのソース行の行番号（10 進数）	__LINE__
__FILE__	ソース・ファイルの名前（文字列定数）	__FILE__
__DATE__	ソース・ファイルの翻訳日付（“Mmm dd yyyy”の形式をもつ文字列定数）	__DATE__
__TIME__	ソース・ファイルの翻訳時間	__TIME__
__STDC__	10 進定数1 （-ansi オプション指定時に定義）	__STDC__ （注1）
__v800 __v800__ __v850 __v850__ __v850e __v850e__ __v850e2 __v850e2__	10 進定数1	__RH850 __RH850__ __v850e3v5 __v850e3v5__ （注2）
__CA850 __CA850__	10 進定数1	__CCRH __CCRH__ （注2）
__CHAR_SIGNED__	10 進定数1（-Xchar オプションで符号つきを指定した場合、および-Xchar オプションを指定しない場合に定義）	__CHAR_SIGNED__ （注2）
__CHAR_UNSIGNED__	10 進定数1（-Xchar オプションで符号なしを指定した場合に定義）	なし
__DOUBLE_IS_32BITS__ _DOUBLE_IS_32BITS	10 進定数1	__DOUBLE_IS_32BITS__ （注2）（注3）
__品種指定名__	10 進定数1（デバイス・ファイル中の「品種指定名」で示される文字列の先頭と末尾に“__”を付けたもの）	なし （注4）
__reg32__	10 進定数1（-reg32オプションを指定した場合、および-regオプションを指定しない場合に定義）	__reg32__ （注2）（注5）
__reg26__	10 進定数1（-reg26オプションを指定した場合に定義）	なし
__reg22__	10 進定数1（-reg22オプションを指定した場合に定義）	__reg22__ （注2）（注6）

注1：-Xansi オプション指定時に定義

注2:値は設定されません。

注3:-Xdbl_size=4 指定時のみ

注4：デバイス・ファイルはサポートしていません。

注5：-Xreg_mode=32オプションを指定した場合に定義します。

注6：-Xreg_mode=22オプションを指定した場合に定義します。

第4章　拡張言語仕様

本章では、CA850 の拡張言語に相当する CC-RH の拡張言語を比較表にて示します。

説明	CA850	CC-RH
アセンブラ命令の記述（注1）	#pragma asm アセンブラ命令 #pragma endasm	#pragma inline_asm 関数名[, 関数名]
	__asm(“アセンブラ命令”);	なし
インライン展開指定	#pragma inline 関数名[, 関数名]	#pragma inline 関数名[, 関数名]
データのセクション割り当て（注2）	#pragma section セクション種別 begin 変数宣言／定義 #pragma section セクション種別 end	#pragma section 属性指定文字 変数宣言／定義 #pragma section default
	#pragma section セクション種別 “セクション名” begin 変数宣言／定義 #pragma section セクション種別 “セクション名” end	#pragma section 属性指定文字 "セクション名" 変数宣言／定義 #pragma section default
関数のセクション割り当て	#pragma text “セクション名” 関数名	#pragma section text "セクション名" 関数定義 #pragma section default
	pragma text “セクション名”	
周辺I/O レジスタ名有効化指定（注3）	#pragma ioreg	なし
割り込み／例外ハンドラ指定（注4）	#pragma interrupt 割り込み要求名 関数名 配置方法 __interrupt 関数定義，または関数宣言	#pragma interrupt 関数名 [(割り込み仕様)]
	#pragma interrupt 割り込み要求名 関数名 配置方法 __multi_interrupt 関数定義，または関数宣言	#pragma interrupt 関数名 (enable=true, [割り込み仕様])
割り込み禁止関数指定	#pragma block_interrupt 関数名	#pragma block_interrupt 関数名
タスク指定	#pragma rtos_task 関数名	なし
構造体パッキング指定	#pragma pack ([1|2|4|8])	#pragma pack ([1|2|4])

注 1:CA850 はＣ言語で記述された関数内にアセンブラ命令を埋め込む際に使用する拡張記述ですが、CC-RH は関数そのものをアセンブラ命令のみとみなして、#pragma inline_asm で宣言したアセンブリ記述関数を呼び出し箇所にインライン展開します。

注 2：指定可能なセクション種別／属性指定文字は CA850 と CC-RH で異なります。

注3:CC-RHはデバイス・ファイルをサポートしていません。周辺I/O レジスタ用のヘッダ・ファイル””iodefine.hをインクルードしてアクセスしてください。

注 4：CA850 は記述した割り込み関数への分岐命令を割り込みハンドラアドレスに自動で配置しますが、CC-RH は割り込み／例外ベクタをお客様自身で定義・配置する必要があります。

第5章　アセンブラ疑似命令

本章では、CA850のアセンブラ疑似命令に相当するCC-RHのアセンブラ疑似命令を比較表にて示します。疑似命令とは、アセンブラが一連の処理を行う際に必要な各種の指示を行うものです。

説明	CA850	CC-RH
セクション定義疑似命令	.bss	.dseg bss
	.const	.cseg const
	.data	.dseg data
	.previous	なし
	.sbss	.dseg sbss
	.sconst	.cseg zconst
	.sdata	.dseg sdata
	.sebss	.dseg ebss
	.section	.section
	.sedata	.dseg edata
	.sibss	.dseg ebss
	.sidata	.dseg edata
	.text	.cseg text
	.tibss .tibss.byte .tibss.word	.dseg tbss4 .dseg tbss5 .dseg tbss7 .dseg tbss8
	.tidata .tidata.byte .tidata.word	.dseg tdata4 .dseg tdata5 .dseg tdata7 .dseg tdata8
	vdbstrtab	なし
	.vdebug	なし
	.vline	なし
シンボル制御疑似命令	.ext_ent_size	なし（注1）
	.ext_func	なし
	.file	.file
	.frame	なし
	.set	.set
	.size	なし

説明	CA850	CC-RH
ロケーション・カウンタ制御疑似命令	.align	.align
	.org	.offset (注2)
領域確保疑似命令	.byte	.db
	.hword	.db2 .dhw
	.shword	.dshw
	.word	.db4 .dw
	.float	.float
	.lcomm	なし
	.space	.ds
	.str	.db
プログラム・リンケージ疑似命令	.globl	.public
	.extern	.extern
	.comm	なし
アセンブラ制御疑似命令	.option reg_mod	$REG_MODE
	.option nomacro	$NOMACRO
	.option macro	$MACRO
	.option data	$DATA
	.option sdata	$SDATA
	.option nowarning	$NOWARNING
	.option warning	$WARNING
ファイル入力制御疑似命令	.binclude	$binclude
	.include	$include
繰り返しアセンブル疑似命令	.irepeat	.irp
	.repeat	.rept

説明	CA850	CC-RH
条件アセンブル疑似命令	.else	$else
	.elseif	$elseif
	.elseifn	$elseifn
	.endif	$endif
	.if	$if
	.ifdef	$ifdef
	.ifn	$ifn
	.ifndef	$ifndef
スキップ疑似命令	.exitm	.exitm
	exitma	.exitma
マクロ疑似命令	.macro	.macro
	.local	.local
	.endm	.endm

注1：再リンク機能はありません。
注 2：CC-RH の.org は絶対アドレス形式セクションの開始を指示する疑似命令です。

第6章　周辺I/Oレジスタ

本章では、CA850 と CC-RH の周辺 I/O レジスタの扱いについて説明します。

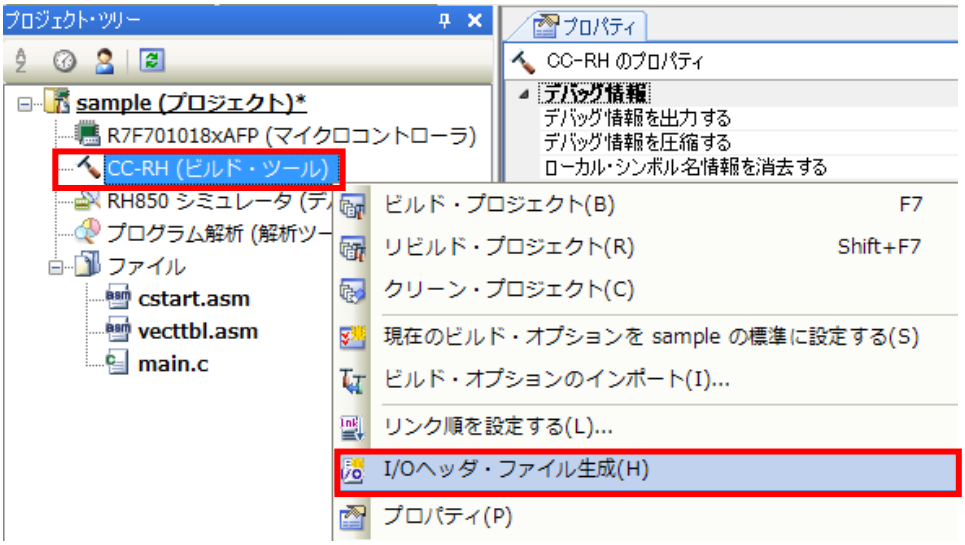
6.1　CA850の周辺I/Oレジスタ

CA850 では、#pragma ioreg 指令を追加することによって C 言語で周辺 I/O レジスタ名を用いてアクセスすることが可能です。レジスタ名と対応アドレス一覧はデバイス・ファイルに登録されており、レジスタ名はアセンブル時にアドレスに変換されます。デバイス・ファイルに登録されているレジスタ名は各マイコンのユーザーズマニュアルに記載されています。

6.2　CC−RHの周辺I/Oレジスタ

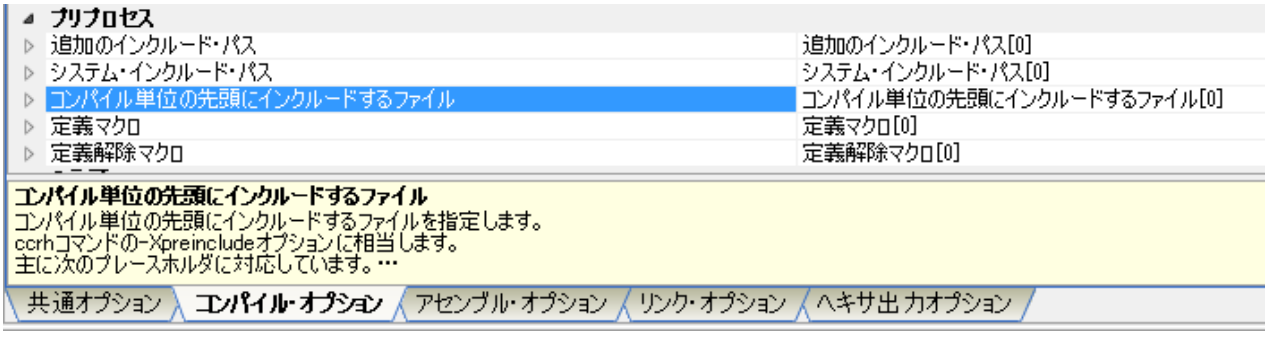
CC-RHはデバイス・ファイルではなく周辺I/Oレジスタ名と対応アドレス一覧を記載した”iodefine.h”をインクルードすることにより周辺I/Oレジスタへのアクセスをサポートします。

CS+では新規プロジェクト作成時に、プロジェクトで指定している該当マイコン用の I/O ヘッダ・ファイル”iodefine.h”を生成し、プロジェクトにソース登録します。I/O ヘッダ・ファイルでは、マイコンが持つレジスタ名と、そのアドレスが定義されています。また、CS+プロジェクト・ツリーの[CC-RH(ビルド・ツール)]ノードを右クリック => [I/O ヘッダ・ファイル生成]を押下して生成させることも可能です。



C 言語プログラム中でレジスタをアクセスする場合、I/O ヘッダ・ファイルをインクルードしてください。

なお、-Xpreinclude オプションの引数に本ファイルを指定することにより、ソースファイル中に#include 指定する必要がなくなります。-Xpreinclude オプションは、[コンパイル・オプション]タブ => [プリプロセス]カテゴリ => [コンパイル単位の先頭にインクルードするファイル]にて指定可能です。本プロパティで該当マイコン用の I/O ヘッダ・ファイルを指定してください。



第7章　　割り込み/例外

本章では、CA850 と CC-RH の割り込み／例外ハンドラについて説明します。

7.1　CA850の割り込み/例外

CA850 は#pragma interrupt 指令により、指定した「割り込み要求名」に対応するハンドラアドレスに「関数名」で指定した関数への分岐命令を埋め込みます。「割り込み要求名」にはデバイス・ファイルに登録されている割り込み要求名を指定します。デバイス・ファイルに登録されている割り込み要求名は、各マイコンのユーザーズマニュアルに記載されています。

```
#pragma interrupt 割り込み要求名 関数名 配置方法
```

また__interrupt 修飾子を付加することによって、当該関数を割り込み関数としてコンパイルします。

```
__interrupt 関数定義，または関数宣言
```

__multi_interrupt 修飾子を付加することによって、当該関数を多重割り込み関数としてコンパイルします。

```
__multi_interrupt 関数定義，または関数宣言
```

例えば V850ES/FJ3 の割り込み要求名”INTP0” のハンドラアドレスは 0xA0 番地です。この場合、以下の#pragma interrupt 指令により、0xA0 番地に”jr _funcint”命令を埋め込みます。また intfunc 関数を割り込み関数としてコンパイルし、割り込み／例外ハンドラとしてのレジスタの退避／復帰処理を出力します。

```
#pragma interrupt INTP0 intfunc
__interrupt void intfunc(void) {
   …;
}
```

7.2 CC−RHの割り込み/例外

CC-RHは#pragma interrupt 指令により、「関数名」で指定した関数を「割り込み仕様」に指定した仕様に従ってコンパイルします。CA850の__interrupt修飾子は不要です。

```
#pragma interrupt 関数名 [割り込み仕様]
```

例えば以下の#pragma interrupt 指令により、intfunc関数を割り込み関数としてコンパイルします。また、指定した割り込み仕様に従って、ctpc／ctpsw／fpepc／fpsr の退避／復帰処理とei／di 命令を出力します。

```
#pragma interrupt intfunc (enable=true, callt=true, fpu=true)
void func (unsigned long eiic)
{
   …;
}
```

なお、CC-RHは割り込み／例外ベクタをお客様自身で定義・配置させる必要があります。

CS+で新規プロジェクトを作成時にソース登録される”boot.asm”では、割り込み／例外ベクタのフォーマットを定義しています。必要に応じてカスタマイズしてください。また、対象マイコンに応じて適切なアドレスに配置させてください。以下、”boot.asm”の割り込み／例外ベクタについて説明します。

a. RESET

下記の定義により、RESET セクションの先頭に「jr32 __start」命令が埋め込まれます。

```
;-----------------------------------------------------------------------
;           exception vector table
;-----------------------------------------------------------------------
            .section "RESET", text
            .align      512
            jr32        __start ; RESET
```

例えば CS+で RH850/F1L 用プロジェクトを新規作成した場合、リンカオプション”-start”により RESET セクションは 0x00 番地に配置指定されています。つまり 0x00 番地に「jr32 __start」命令が埋め込まれます。

b. 直接ベクタ方式の例外／割り込み

ハンドラアドレスの基準位置は、RBASEレジスタ、またはEBASEレジスタで示されるベース・アドレスに例外要因のオフセットを加算した値を使用します。いずれをベース・アドレスとして利用するかは、PSW.EBVビットによって選択します。下記の定義では、RBASEをベース・アドレスであるものとして、RESETの直後から割り込み／例外ハンドラを配置しています。

```
            .section "RESET", text
            .align      512
            jr32        __start ; RESET

            .align      16
            jr32        _Dummy ; SYSERR

            .align      16
            jr32        _Dummy
```

”boot.asm”では、各割り込み／例外ハンドラの対応するオフセット位置に、ダミー関数_Dummy に分岐する命令を配置しています。_Dummy は自分自身への分岐を繰り返すルーチンです。必要に応じてカスタマイズしてください。

カスタマイズ対象とする例外／割り込みに対応するオフセット位置の「_Dummy」を「_割り込み関数名」に変更してください。また、#pragma interrupt 指令によって割り込み関数を定義してください。以下は例外”SYSERR”発生時に割り込み関数 func を実行する場合の記述例です。

```
        .section "RESET", text
        .align    512
        jr32      __start ; RESET
                                  「__Dummy」を「_割り込み関数名」に変更
        .align    16
        jr32      _func ; SYSERR

        .align    16
        jr32      _Dummy
        ・・・
```

```
#pragma interrupt func (priority=SYSERR, callt=true, fpu=true)
void func1(unsigned long feic)
{
   …;
}
```

c. テーブル参照方式の例外／割り込み

割り込みの拡張仕様として、テーブル参照方式による割り込みを指定できます。

直接ベクタ方式では、EIレベル割り込みのハンドラアドレスはそれぞれの割り込み優先度ごとに1つであり、複数の同一優先度を示す割り込みチャネルは同じ割り込みハンドラアドレスへ分岐します。しかし、アプリケーション上、割り込みハンドラごとに異なるコード領域を利用したい場合などを想定し、テーブル参照方式を実装しています。

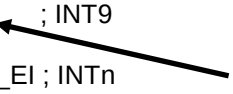
```
        .section "EIINTTBL", const
        .align    512
        .dw       #_Dummy_EI ; INT0
        .dw       #_Dummy_EI ; INT1
        .dw       #_Dummy_EI ; INT2
        .rept     512 - 3
        .dw       #_Dummy_EI ; INTn
        .endm
```

”boot.asm”では、EIINTTBL セクションにテーブル参照方式の例外／割り込みテーブルを定義しています。CS+で RH850/F1L 用プロジェクトを新規作成した場合、リンカオプション”-start”により RESET セクションの直後に配置指定されています。

EIINTTBL セクションの先頭から 4 の倍数の領域にダミー関数_Dummy_EI の配置アドレスが埋め込まれています。これにより、割り込み優先度 n のテーブル参照方式の例外／割り込みが発生した場合、_Dummy_EI に分岐します。_Dummy_EI は自分自身への分岐を繰り返すルーチンです。必要に応じてカスタマイズしてください。

カスタマイズ対象とするチャネルに対応するオフセット位置の「#_Dummy_EI」を「#_割り込み関数名」に変更してください。また、#pragma interrupt 指令によって割り込み関数を定義してください。以下はチャネル 9”EIINT9”発生時に割り込み関数 func を実行する場合の記述例です。

```
        .section "EIINTTBL", const
        .align    512
        .dw       #_Dummy_EI ; INT0
        .dw       #_Dummy_EI ; INT1
        .dw       #_Dummy_EI ; INT2
        .dw       #_Dummy_EI ; INT3
        .dw       #_Dummy_EI ; INT4
        .dw       #_Dummy_EI ; INT5
        .dw       #_Dummy_EI ; INT6
        .dw       #_Dummy_EI ; INT7
        .dw       #_Dummy_EI ; INT8
        .dw       #_func     ; INT9
        .rept     288 - 10
        .dw       #_Dummy_EI ; INTn
```



「#_Dummy_EI」を
「#_割り込み関数名」に変更

```
#pragma interrupt func (channel=9 enable=true, callt=true, fpu=true)

void func (unsigned long eiic)
{
   …;
}
```

なお、直接ベクタ方式の例外／割り込みが RH850 のデフォルトであるため、テーブル参照方式に変更する場合は割り込み制御レジスタの値を変更する必要があります。

第8章　ROM化

初期値付き変数のデータはROMに配置しておき、リセット後の実行時にRAMにコピーする必要があります。この一連の処理をROM化といいます。ROM化の方法はCA850とCC-RHで異なります。本章では、CA850とCC-RHのROM化方法について説明します。

8.1　CA850のROM化方法

CA850の場合、初期値あり変数が配置されるセクション（.sdata／.data セクション）はデフォルトでROM化対象としています。初期値データはROM化用領域確保コード（rompcrt.o）を配置したROM上のアドレスに配置されており、_rcopy関数を使用してROMからRAMにコピーします。コピー先のアドレス（sdata／.dataセクションのアドレス）はリンク・ディレクティブ・ファイル（*.dir）で指定します。

以下はROM化時にrompcrt.oを使用した場合の、コピー関数を呼び出すCソース例です。

```
int _rcopy(unsigned long *, long);
extern unsigned long _S_romp;

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

「_S_romp」はROM化用領域確保コード・ファイルrompcrt.oで定義しているシンボルであり、「&_S_romp」はROMに格納されている初期値データの先頭アドレスです。リンカが自動的に決定します。

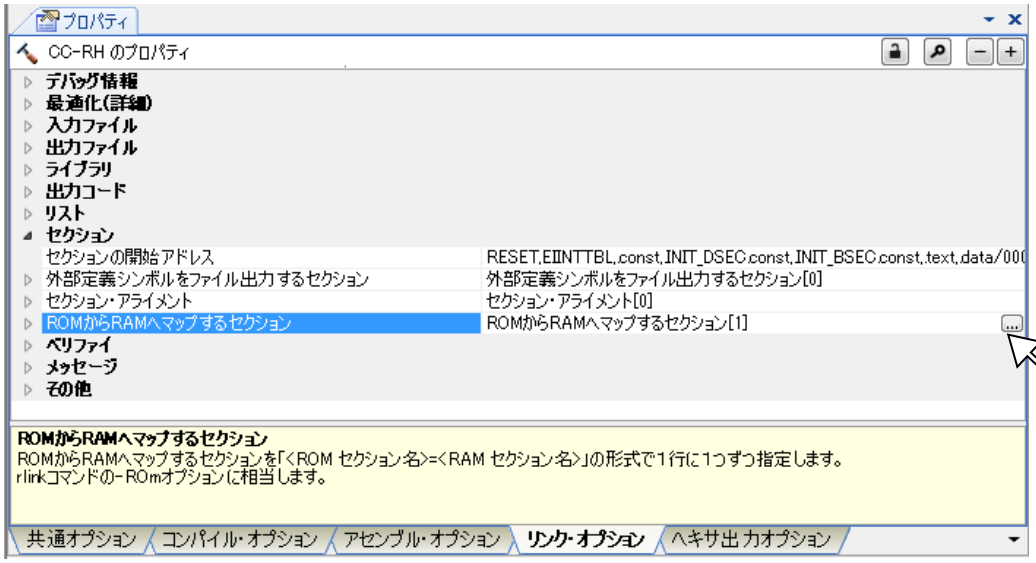
8.2 CC-RHのROM化方法

a. ROM化指定

CC-RHの場合、ROM化対象とするセクションをリンカの-romオプションにより指定します。*<ROMセクション名>*がROM化対象のセクションとなります。リンカの-startオプションによって、*<ROMセクション名>*で指定したセクションはROMに配置指定し、*<RAMセクション名>*で指定したセクションはRAMに配置指定してください。

```
-rom=<ROM セクション名>=<RAM セクション名>
```

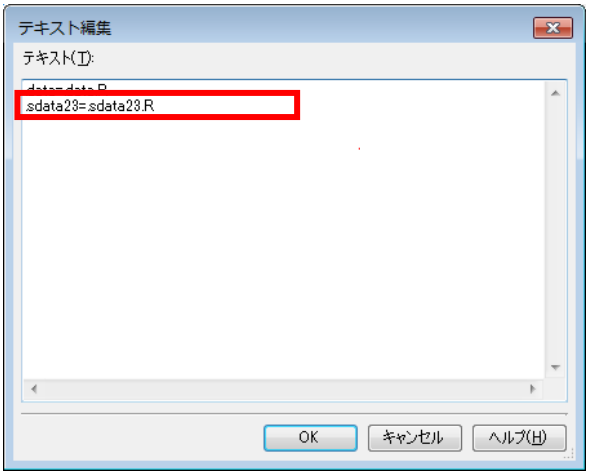
CS+では[リンク・オプション]タブ => [セクション]カテゴリ => [ROMからRAMへマップするセクション] にて右端の[...]ボタンを押下し、ROMからRAMへコピーするセクションを「*<ROMセクション名>=<RAMセクション名>*」の形式で、1行に1つづつ指定してください。



CS+で新規プロジェクトを作成した場合、以下のオプションがデフォルトで指定されています。

```
-rom=.data=.data.R
```

.dataセクション以外に初期値あり変数が配置されるセクションを追加した場合、-romオプションによりROM化対象として追加指定してください。以下は.sdata23セクションを追加でROM化対象とする場合（RAMセクション名を.sdata.Rとする場合）の指定例です。



b. 初期化テーブルの定義

CC-RHでは_INITSCT_RH関数を使用してROM化したデータをROMからRAMにコピーします。CS+で新規プロジェクトを作成時にソース登録されるスタートアップ・ルーチン”cstartm.asm”では、初期値あり変数のデータをコピーする際に使用する初期化テーブルを以下のような定義しています。

```
;-----------------------------------------------------------------------
;       section initialize table
;-----------------------------------------------------------------------
        .section  ".INIT_DSEC.const", const
        .align    4
        .dw       #__s.data,        #__e.data,        #__s.data.R
```

初期化テーブルは.INIT_DSEC.const セクションに配置指定されており、.dataセクションの先頭アドレス->.dataセクションの終端アドレス->.data.Rセクションの先頭アドレスの順番でそれぞれ4バイトづつ領域が確保されています。

なお、セクション名の頭に”__s”を付けることで、そのセクションの先頭アドレスを値として持つ予約シンボルとなります。同様にセクション名の頭に”__e”を付けることで、そのセクションの終端アドレスを値として持つ予約シンボルとなります。初期化テーブルへの追加はこの予約シンボルを使用頂くことを推奨します。

初期値あり変数が配置されるセクションを追加した場合は、この初期化テーブルにもそれぞれセクションの先頭アドレス・終端アドレスを追加してください。

```
;-----------------------------------------------------------------------
;       section initialize table
;-----------------------------------------------------------------------
        .section  ".INIT_DSEC.const", const
        .align    4
        .dw       #__s.data,        #__e.data,        #__s.data.R
        .dw       #__s.sdata23,     #__e.sdata23,     #__s.sdata23.R
```

CC-RHでは、_INITSCT_RH関数を使用して、初期値なし変数が配置されるセクションをゼロ初期化することも可能です。スタートアップ・ルーチン”cstartm.asm”でゼロ初期化テーブルを定義しています。

```
        .section  ".INIT_BSEC.const", const
        .align    4
        .dw       #__s.bss,         #__e.bss
```

ゼロ初期化テーブルは.INIT_BSEC.const セクションに配置指定されており、.bssセクションの先頭アドレス->.bssセクションの終端アドレスの順番でそれぞれ4バイトづつ領域が確保されています。.bssセクション以外に初期値なし変数が配置されるセクションを追加した場合は、同様の手順でそれぞれのアドレスを追加してください。

c. コピー関数の呼び出し

スタートアップ・ルーチン”cstartm.asm”で_INITSCT_RH 関数を呼び出しています。この処理により、各テーブルで定義したセクションを初期化します。

```
        mov       #__s.INIT_DSEC.const, r6
        mov       #__e.INIT_DSEC.const, r7
        mov       #__s.INIT_BSEC.const, r8
        mov       #__e.INIT_BSEC.const, r9
        jarl32    __INITSCT_RH, lp            ;   initialize RAM area
```

第9章　　セクション配置

本章では、CA850とCC-RHのセクション配置方法を説明します。

9.1　CA850のセクション配置

CA850はリンク・ディレクティブ・ファイル（*.dir）というファイルにセクションの配置アドレスを記述し、リンク・ディレクティブ・ファイルをリンカに入力することでセクションの配置アドレスを指定します。以下はCA850のリンク・ディレクティブ・ファイルのフォーマットです。

```
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 {

        出力セクション名=$セクションタイプ　?セクション属性　入力セクション名;
        出力セクション名=$セクションタイプ　?セクション属性　入力セクション名;
        …
} ;
```

下記の記述により、.const セクションを0x1000番地から配置し、.const セクションの終端から上位方向に.pro_epi_runtime =>.text セクションを配置します。また、0xfedf6000番地から上位方向に.data => .sdata => .sbss => .bss セクションを配置します。

```
CONST:!LOAD ?R V0x1000 {
        .const = $PROGBITS ?A .const ;
};

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

DATA:!LOAD ?RW V0xfedf6000 {
        .data = $PROGBITS ?AW .data ;
        .sdata = $PROGBITS ?AWG .sdata ;
        .sbss = $NOBITS ?AWG .sbss ;
        .bss = $NOBITS ?AW .bss ;
};
```

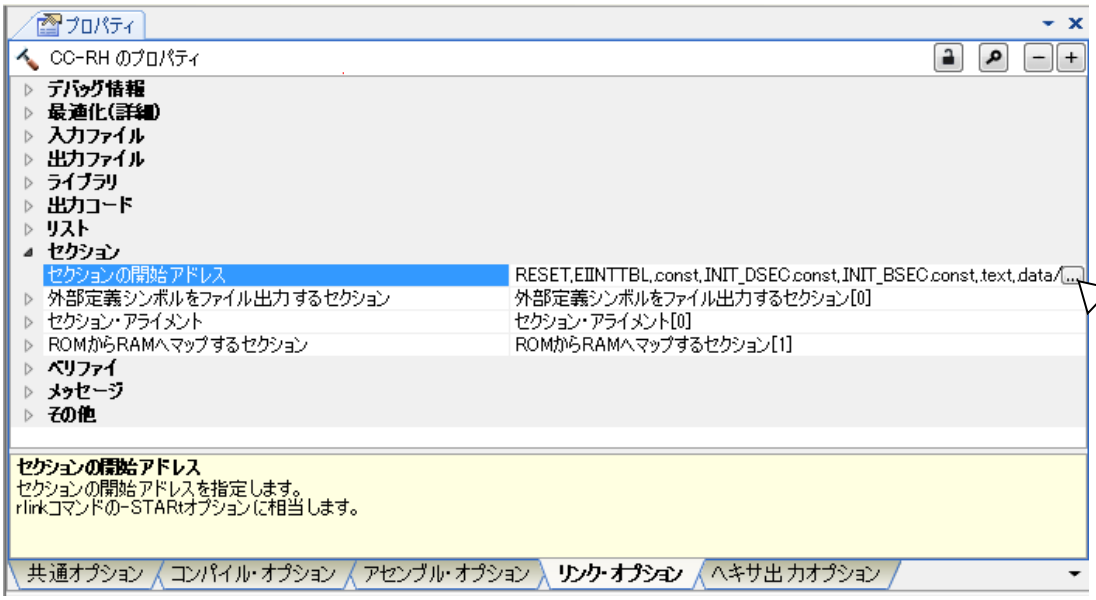
9.2 CC−RHのセクション配置

CC-RHはリンカオプション”-start”によりセクションの配置アドレスを指定します。CA850のリンク・ディレクティブ・ファイルのようなファイルで指定する形式ではありません。以下はCC-RHの-startオプションの指定例です。

```
-start=RESET,EIINTTBL,.const,.INIT_DSEC.const,.INIT_BSEC.const,.text,.data/00000000,
.data.R,.bss,.stack.bss/FEDE0000
```

上記オプションにより、RESETセクションを0x00番地から配置し、RESETセクションの終端から上位方向にEIINTTBL => .const => .INIT_DSEC.const => .INIT_BSEC.const => .text => .data セクションを配置します。また、0xFEDE0000番地から上位方向にdata.R => .bss => .stack.bss セクションを配置します。

CS+ではGUI上から指定することが可能です。 [リンク・オプション]タブ => [セクション]カテゴリ => [セクションの開始アドレス] にて右端の[...]ボタンを押下してください。



セクション設定ダイアログが起ち上がりますので、このダイアログ上からの操作により配置アドレスやセクションを追加・変更することが可能です。