Development Environment Migration Guide
Migrating from V850 to RH850 (Compiler Guide)

R20UT4014EJ0100
Rev.1.00
2017.3.16

This document describes the main points to note in migration from the V850 compiler (CA850) to the RH850 family compiler (CC-RH).

Contents

Chapter 1 Option

This chapter gives comparison tables listing correspondences between options for the CC-RH and CA850 compilers.

CC-RH is case-sensitive to characters for compiler and assembler options but not to those for linker options. CA850 is case-sensitive to characters in all types of options.

1.1 Compiler Options

Classification	Description	CA850(ca850.exe)	CC-RH(ccrh.exe)
Version/help display/operation status	Outputs the version information of the C compiler to the standard error output.	-V	-V
	Outputs option descriptions to the standard error output.	-help	-h
	Outputs the execution status of the C compiler to the standard error output in detail.	-v	None.
Output file specification	Specifies where an intermediate language file is to be saved.	-Fic	None.
	Specifies where an object file is to besaved.	-Fo	-Xobj_path
	Specifies where an assembly language file is to be saved.	-Fs	-Xasm_path
	Specifies where an assemble list file is to be saved.	-Fv	-Xasm_option=-Xprn_path
	Specifies the output file.	-o	-o
	Specifies the work folder.	-temp	None.
Controlling source debugger	Prohibits replacing the ld.w/ld.h and st.w/st.h instructions with 1-bit manipulation instructions.	-Xno_word_bitop	None. (note1)
	Outputs symbol information for the source debugger.	-g	-g
Device specification	Treats the memory space as having 256 MB.	-X256M	None.
	Sets the higher address of the programmable peripheral I/O register.	-Xbpc	None.
	Embeds the magic number common to V850 core.	-cn	None.
	Embeds the magic number common to V850Ex core.	-cnv850e	
	Embeds the magic number common to V850E2 core.	-cnv850e2	
	Specifies the target device.	-cpu	None. (note2)
	Specifies the folder to search device files.	-Xdev_path	
Compiler control specification	Outputs the assembler source file without executing any modules after the assembler.	-S	-S
	Outputs an assemble list.	-a	-Xasm_option=-Xprn_path
	Outputs the object file without starting the linker.	-c	-c
	Executes the only front end, generates an .ic file, and then terminates processing.	-m	None.
ROMization control	This option is necessary when creating a ROMization object.	-Xr	-Xlk_option=-rom

Classification	Description	CA850(ca850.exe)	CC-RH(ccrh.exe)
Preprocessor processing setting	Includes source program comments in the preprocessing output.	-C	-Xprerocess=comment
	Assumes that #define is entered before the C source program.	-D	-D
	Executes preprocessing only for a C source program and outputs the results to the standard output.	-E	None.
	Specifies the folder to search the header file of the C source program.	-I	-I
	Executes preprocessing only for a C source program and outputs the results to a file.	-P	-P
	Assumes that #undef is entered before the C source program.	-U	-U
	Assumes that .set is entered before the assembler source.	-Wa,-D	-D
	Specifies the folder to search the header file of the assembler source file.	-Wa,-I,	-I
	In addition to ordinary comments, interprets all characters that appear after "//" and before the end of the line as comments.	-Xcxxcom	Always in effect
	Outputs a warning message in response to initialization of a pointer type external variable which uses a variable address that is not an automatic variable or which uses a function address.	-Xd	None.
	Specifies the upper limit for the number of macro identifiers.	-Xm	None. (note3)
	Replaces a trigraph sequence.	-t	Always in effect
Memory saving during compilation	Reduces the memory capacity used in the pre-optimizer phase during compiling.	-Wp,-D	None.
	Reduce the memory capacity used in the machine dependent optimization phase during compiling.	-Wi,-D	
Error output specification	Adds and saves error messages to the file.	+err_file	None.
	Overwrites and saves error messages to the file.	-err_file	-Xerror_file
	Specifies the maximum number of error messages to be output.	-err_limit	None. (note4)
Expansion function specification	Enables the expansion functions compatible with the 78K microcontrollers C compiler CC78Kx.	-cc78k	None. (note5)
Optimization	This is the optimize for debugging option.	-Od	-Onothing
	This is the default optimization option.	-Ob	-Odefault
	This is the standard optimization option.	-Og	None.
	This is the Level 1 advanced optimization.	-O	None.
	This is the Level 2 advanced optimization option (object size precedence).	-Os	-Osize
	This is the Level 2 advanced optimization option (execution speed precedence).	-Ot	-Ospeed

Classification	Description	CA850(ca850.exe)	CC-RH(ccrh.exe)
Target code optimization	Analyzes the data flow strictly and perform the most advanced optimization.	-Wi,-O4	None.
	Prevents optimization that allows branch destination labels to be aligned.	-Wi,-P	This optimization is suppressed by default (note6)
File merging	Merges the files when two or more files are specified at the same time.	-Om	-Xmerge_files
Inline expansion optimization control	Restricts the stack size for a function subject to inline expansion in the intermediate language so that inline expansion is not performed for the large value.	-Wp,-G	None.
	Restricts the intermediate language size for a function subject to inline expansion so that inline expansion is not performed for the large value.	-Wp,-N	-Oinline_size (note7)
	Performs inline expansion of a static function that is referenced only once unconditionally.	-Wp,-S	None.
	Outputs function information to the standard output or additionally outputs to the file.	-Wp,-l	None.
	Performs inline expansion of only a function for which #pragma inline is specified.	-Wp,-inline	-Oinline=1
	Suppresses inline expansion of all functions, including the function for which #pragma inline is specified.	-Wp,-no_inline	-Oinline=0
	Deletes unnecessary functions from the functions called from an entry function after inline expansion.	-Wp,-r	None.
Loop expansion optimization control	Expands a loop the specified times using "for" and "while".	-Wo,-Ol	-Ounroll
	Expands a loop by fixing the number of times of expanding the loop.	-Wo,-Xlo	None.
strcpy, strcmp expansion	Sets a 4-byte alignment condition for arrays and structures and performs inline expansion of strcpy() or strcmp() function calls.	-Xi	-Xinline_strcpy (note8)
External variable sort	Rearranges external variables starting from the largest alignment size.	-Wo,-Op	None.
Branch instruction control	Arranges and outputs branch instructions, giving precedence to the code size.	-Wo,-XFo	None.
Register use control	Allocates the specified external variable to the specified register.	-r	None.
	Limits the number of registers used by the C compiler.	-reg	-Xreg_mode (note9)
	Uses the mask register function.	-Xmask_reg	None. (note10)
Prologue/epilogue processing control	Specifies whether or not to perform prologue/epilogue processing of the function based on runtime library function calls.	-Xpro_epi_runtime	None. (note11)

Classification	Description	CA850(ca850.exe)	CC-RH(ccrh.exe)
signed/unsigned control	Specifies whether int type bit fields that do not indicate the type specifier are handled as signed or unsigned.	-Xbitfield	None.
	Specifies whether char type that do not indicate the type specifier are handled as signed or unsigned.	-Xchar	None. (note12)
	Specifies which integer type the enumeration type matches.	-Xenum_type	-Xenum_type (note13)
Variable placement control	Allocates data of less than the specified bytes to the .sdata or .sbss section.	-G	-Xsection (note14)
	Allocates const attribute data and character string literals to the .sconst section.	-Xsconst	
	Outputs the frequency information file for the variables used by the section file generator.	-Xcre_sec_data	-Omap -Osmap (note15)
		-Xcre_sec_data_only	
	Specifies the name of the section file that is used to specify section allocation of data when the C compiler is activated.	-Xsec_file	
Switch-case statement output code control	Specifies a mode in which the code of a switch statement is to be output.	-Xcase	-Xswitch
	Generates one 4-byte branch table per case label in a switch statement.	-Xword_switch	None. (note16)
Structure packing control	Specifies indirect address access to a structure in byte units.	-Xbyte	None.
	Specifies alignment of structure members.	-Xpack	-Xpack
Far jump output control	Uses jmp directive to branch to the specified function.	-Xfar_jump	-Xfar_jump
	Uses the jmp instruction for an ordinary interrupt function defined in C language.	-Xj	None. (note17)
Comment output	Outputs the C source program as a comment to the assembler source file.	-Xc	-Xpass_source
ANSI standard	Uses runtime library, without using the mulh and divh directives for integers corresponding to data that is 16 bits or less.	-Xe	None.
	Treats tentative definition of variables as definition.	-Xdefvar	None.
	Makes C compiler processing comply strictly with the ANSI standard and outputs an error or warning for a specification that violates the standard.	-ansi	-Xansi

Classification	Description	CA850(ca850.exe)	CC-RH(ccrh.exe)
Library specification	Specifies the folder to search libraries.	-L	None. (note18)
	Specifies the startup module to be used when startup goes as far as the linker.	-R	None.
	Specifies the archive file that is referenced by the linker.	-l	-Xlk_option=-library=*file*
Warning message control	Specifies the level, output, and suppression of a warning message.	-w	None.
	Outputs a warning message of the specified number.	-won	None.
	Suppresses a warning message of the specified number.	-woff	-Xno_warning
Command file specification	Handles the specified file as a command file.	@	@
CPU bug patch	Specifies the -p option for the assembler for an assembler source file output by the C compiler to output a code corresponding to a CPU fault.	-Xv850patch	-Xpatch
Each module	Specifies options to each module.	-W	-Xasm_option -Xlk_option (note19)
Other	Performs advanced optimization.	+Oc	None.

Notes:

*1: Regarding rules for the generation of bit manipulation instructions, refer to section 11.3, Controlling the Output of Bit Manipulation Instructions (in V1.05.00 and later versions), in the CC-RH Compiler User's Manual (R20UT3516EJ0102).

*2: The CC-RH compiler does not support device files.

*3: The upper limit depends on the amount of memory on the host machine on which the program is to run.

*4: There is no upper limit on the number of error messages to be output.

*5: There is an option "-Xcheck=shc" for checking C source files which have been coded for SHC.

*6: Alignment is enabled by specifying the -Xalign4=all option.

*7: This is specified with the degree to which the code size can be increased as a percentage by inline expansion.

*8: The memcpy and memset functions are also targets for inline expansion.

*9: CC-RH does not have a 26-register mode.

*10: The RH850 does not have mask register functions since zero extension is handled on the hardware side.

*11: Runtime library function calls are not always used in prologue and epilogue processing for functions.

*12: Variables without signs that are simply declared as char are always handled as signed.

*13: Integer types cannot be specified; they are automatically determined.

*14: This option was added in CC-RH V1.02.00; it collectively changes the default allocation of variables to sections.

*15: There are no section files. When the -Omap or -Osmap option is specified, variables that are frequently accessed are optimized for EP-relative one-instruction accesses.

*16: The widths of branch tables are automatically determined.

*17: The user must define the vector table.

*18: The library specified with the -library option is preferentially linked. If there is an unresolved symbol, the library is found in order of the environment variables HLNK_LIBRARY1, HLNK_LIBRARY2, and HLNK_LIBRARY3.

*19: -Xasm_option and -Xlk_option lead to the passing of arguments to the assembler and linker, respectively.

1.2 Assembler Options

Classification	Description	CA850(as850.exe)	CC-RH(asrh.exe)
File	Generates an assemble list.	-a	-Xprn_path (note1)
	Adds and saves error messages to the file.	+err_file	None.
	Overwrites and saves error messages to the file.	-err_file	-Xerror_file
	If the -a option is specified, an assemble list generated is saved.	-l	-Xprn_path
Assembler	Specifies the macro name to be defined.	-D	-D
	Generates a machine language instruction on the assumption that the data that is less than the specified bytes is allocated to sections with the sdata or sbss attribute in response to external label access.	-G	None. (note2)
	Specifies the folder where the file specified by the file input quasi directive is given precedence to searching.	-I	-I
	Generates an object file that includes information noting use of the mask register function.	-m	None. (note3)
	Performs optimization that rearranges instructions to avoid register/flag hazards.	-O	None.
	Outputs the execution status of the assembler to the standard error output in detail.	-v	None.
	Specifies the level, output, and suppression of a warning message.	-w	None.
	Specifies far jump for branch instructions (jarl, jr) that do not include 22/32.	-Xfar_jump	-Xasm_far_jump

Classification	Description	CA850(as850.exe)	CC-RH(asrh.exe)
Device	Treats the memory space as having 256 MB.	-X256M	None.
	Sets the higher address of the programmable peripheral I/O register.	-bpc	None.
Warning message control	Suppresses a warning message of the specified number.	-woff	-Xno_warning
Other	Embeds the magic number common to V850 core.	-cn	None.
	Embeds the magic number common to V850Ex core.	-cnv850e	
	Embeds the magic number common to V850E2 core.	-cnv850e2	
	Specifies the target device.	-cpu	None. (note4)
	Specifies the folder where device files are stored.	-F	
	Outputs debug information.	-g	-g
	Specifies the name of the object file to be assembled and output.	-o	-o
	Outputs code that avoids CPU faults.	-p	None.
	Outputs the version information of the assembler to the standard error output.	-V	-V
	Performs assembly processing on the flash/external ROM side.	-zf	None.
	Handles the specified file as a command file.	@	@

Notes:

*1: A file is not output to the standard output stream but to an assembler listing file.

*2: Generation of machine language instructions cannot be controlled with an option; it is solely determined by the source code.

*3: The RH850 does not have mask register functions since zero extension is handled on the hardware side.

*4: The CC-RH compiler does not support device files.

1.3 Linker Options

Classification	Description	CA850(ld850.exe)	CC-RH(rlink.exe)
Input file	Performs linking according to the specified link directive in link directive file.	-D	-start (note1)
	Selects the compatibility of the format of the link directive file with old versions.	-Xolddir	
Output file	Adds and saves error messages to the file.	+err_file	None.
	Overwrites and saves error messages to the file.	-err_file	None. (note2)
	Specifies the name of the object file to be generated.	-o	-output
	Outputs a link map that indicates allocation of the input and output sections to the memory space.	-m	-list -show
	Outputs a link map that indicates allocation of the input and output sections to the memory space in the format of products older than CA850 Ver. 2.60.	-mo	-list -show
Library	Searches the archive file (library file) specified by the -l option from the specified folder, standard folder in that order.	-L	None. (note3)
	Links the standard library of the compiler (libc.a).	-lc	-library (note4)
	Links the mathematical library of the compiler (libm.a).	-lm	
	References the specified archive file when resolving an unresolved external symbol reference.	-l	-library
Flash	Generates an object file for the flash/external ROM relink function using the value specified as the start address value of the branch table.	-ext_table	None. (note5)
	Generates the flash area object file from the specified object file as the boot area object file.	-zf	
Device	Treats the memory space as having 256 MB.	-X256M	None.
	Sets the security ID of an on-chip flash memory device.	-Xsid	None.
	Suppress the option byte that is generated by default.	-Xob=none	None. (note6)

Classification	Description	CA850(ld850.exe)	CC-RH(rlink.exe)
Linker	Outputs as the standard output the information that can be used as a yardstick for the sdata/sbss data allocation option that is specified for the ca850 and as850.	-A	None.
	Performs linking in the 2-pass mode.	-B	None.
	Outputs a warning message, not an error message, and continues linking if an illegality is found during relocation processing.	-E	-change_message (note7)
	Outputs a message for all multi-defined external symbols and stops link processing.	-M	None.
	Does not check the size and alignment condition when linking an external symbol.	-T	None.
	Controls checking when the internal ROM/RAM overflows.	-Ximem_overflow=warning	-cpu (note8)
	Regards the specified symbol value as the entry point address value for the object file to be generated.	-e	-entry
	Specifies the filling value for align holes between sections of the generated object.	-f	-space
	Checks whether or not the files that use the mask register function are mixed with files that do not use this function.	-mc	None. (note9)
	Outputs detailed information when register modes are mixed for all input object files.	-rc	None. (note10)
	Re-references the library file specified by the -l option.	-rescan	Re-reference is the default behavior
	Does not check for the allocation to the internal ROM area.	-romless	None. (note11)
	Generates an object file in which the debug information, line number information, and global pointer table have been removed.	-s	-nodebug -strip
	Does not check the size and alignment condition of the symbol when linking an undefined external symbol.	-t	None.
	Outputs the execution status of the linker in detail.	-v	None.
	Does not output a warning messages.	-w	-change_message=information=<*number*> -nomessage (note12)

Classification	Description	CA850(ld850.exe)	CC-RH(rlink.exe)
Other	Searches a device file from the specified folder.	-F	None. (note13)
	Reads the device file for the target device specified.	-cpu	
	Outputs the version information of the C compiler to the standard error output.	-V	None. (note14)
	Outputs option descriptions to the standard error output.	-help	
	Checks whether or not the old function calling and the calling specification of the current version are mixed for all input object files.	-fc	None.
	References the library for a mask register function.	-mask_reg	None. (note9)
	Generates a relocatable object file.	-r	-form=relocate
	Generates a relocatable object file in the old mapping mode (CA850 Ver.2.30 or earlier).	-ro	
	References the corresponding register mode library.	-reg	None. (note15)
	Handles the specified file as a command file.	@	-subcommand

Notes:

*1: Linkage directive files are not supported.

*2: Error messages are displayed in the linkage map file.

*3: The library specified with the -library option is preferentially linked. If there is an unresolved symbol, the library is found in order of the environment variables HLNK_LIBRARY1, HLNK_LIBRARY2, and HLNK_LIBRARY3.

*4: The configuration of the standard libraries differs between V1.02.00 and earlier versions and versions later than that. In addition, the names of libraries that are specified as arguments also differ according to library functions or conditions imposed by options in use. For details, section 7.1, Supplied Libraries, in the CC-RH Compiler User’s Manual.

*5: No relinking function is provided.

*6: Option bytes cannot be specified with CC-RH. Specify OPTB0 as a [Flash Options] property on the [Flash Options Setting] tabbed page.

*7: The -change_message option can change an error message to a warning message and continue processing.

*8: Specify an address as the argument. Since an error message is output by default, change the error number to a warning number by using -change_message.

*9: The RH850 does not have mask register functions since zero extension is handled on the hardware side.

*10: Input of a file having a different register mode leads to an E0562408 error.

*11: Allocation to the internal ROM area is not checked by default.

*12: This option can be hidden by using the -change_message option to change a warning message to an information message and using the -nomessage option to suppress information messages.

*13: The CC-RH compiler does not support device files.

*14: Display in the same way as is selected by these options is obtained by entering the rlink command and pressing the [ENTER] key on the command line.

*15: Standard libraries are not provided for individual register modes. Reference is always to the standard library generated by -Xreg_mode=common.

1.4 Hex Converter Options

Classification	Description	CA850(hx850.exe)	CC-RH(rlink.exe)
File	Adds and saves error messages to the file.	+err_file	None.
	Overwrites and saves error messages to the file.	-err_file	None. (note1)
	Outputs the hex-converted result to the specified file.	-o	-output
Format	Regards the specified value as the maximum block length.	-b	-byte_count (note2)
	Specifies the offset of the address to be output.	-d	None.
	Specifies the hex format.	-f	-form
	Converts and outputs code in the specified section.	-I	-output (note3)
	Converts and outputs a symbol table.	-S	None.
	Converts into hex format and outputs all the codes in the area specified by the specified address to the specified size.	-U	-output (note4) -space
	When converts and outputs the symbol table, also converts and outputs local symbols.	-x	None.
	Disables use of the information of the internal ROM area defined by the device file when the -U option is specified.	-rom_less	None. (note4) (note5)
	Generates as many null characters (¥0) as the size of a section for a section with the section type NOBITS and section attribute A.	-z	None.
Other	Searches a device file from the specified folder.	-F	None. (note5)
	Outputs the version information of the C compiler to the standard error output.	-V	None. (note6)
	Handles the specified file as a command file.	@	-subcommand

Notes:

*1: Error messages are displayed in the linkage map file.

*2: This option is only valid when Intel HEX files are specified.

*3: A section *section* is specified in the format -output=*file*=*section*.

*4: An address *address* is specified in the format -output=*file*=*address1*-*address2*.

*5: The CC-RH compiler does not support device files.

*6: For CC-RH, the linker (rlink.exe) handles conversion of HEX files. Information on the rlink version is displayed by entering the rlink command on the command line and pressing the [ENTER] key.

Chapter 2 Intrinsic Functions

This chapter gives a comparison table listing correspondences between intrinsic functions for the CC-RH and CA850 compilers.

If an intrinsic function which is for CA850 rather than CC-RH is called, CC-RH compiles it as an ordinary function. If there is no definition, an error will occur during linking.

Instruction	Description	CA850	CC-RH
di	Disables interrupts	void __DI(void);	void __DI(void);
ei	Enables interrupts	void __EI(void);	void __EI(void);
nop	No operation	void __nop(void);	void __nop(void);
halt	Stops the processor	void __halt(void);	void __halt(void);

satadd	Saturated addition	long a, b;
		long __satadd(a, b);
			long a, b;
			long __satadd(a, b);

satsub	Saturated subtraction	long a, b;
		long __satsub(a, b);
			long a, b;
			long __satsub(a, b);

bsh	Halfword data byte swap	long a;
		long __bsh(a);
			long a;
			long __bsh(a);

bsw	Word data byte swap	long a;
		long __bsw(a);
			long a;
			long __bsw(a);

hsw	Word data halfword swap	long a;
		long __hsw(a);
			long a;
			long __hsw(a);

sxb	Byte data sign extension	char a;
		long __sxb(a);
			None.

sxh	Halfword data sign extension	short a;
		long __sxh(a);
			None.

mul	Instruction that assigns the higher 32 bits of multiplication result to a variable using mul32 instruction	long a, b;
		long __mul32(a, b);
			long a, b;
			long __mul32(a, b);

mulu	Instruction that assigns the higher 32 bits of unsigned multiplication result to a variable using mul32u instruction	unsigned long a, b; unsigned long __mul32u(a,b);	unsigned long a, b; unsigned long __mul32u(a, b);

sasf	Flag condition setting with logical left shift	long a; unsigned int b;
		long __sasf(a, b);
			None.

—	Controls interrupt level (note1)	int NUM;
		void __set_il(NUM, *"interrupt-request name"*);
			int NUM;
			void* ADDR;
			void __set_il_rh(NUM, ADDR);

note1 : In the CA850, a character string defined in the device file should be specified as “interrupt-request name". In the CC-RH, specify the address of the interrupt control register as ADDR.

Chapter 3 Predefined Macros

This chapter gives a comparison table listing correspondences between predefined macros for the CC-RH and CA850 compilers.

CA850 Macro Name	CA850 Definition	CC-RH Macro Name
__LINE__	Line number of source line at that point (decimal).	__LINE__
__FILE__	Name of assumed source file (character string constant).	__FILE__
__DATE__	Decimal constant 1 (defined when -ansi option is specified)	__DATE__
__TIME__	Translation time of source file	__TIME__
__STDC__	Decimal constant 1. (Defined when the -Xansi option is specified)	__STDC__ (note1)
__v800 __v800__ __v850 __v850__ __v850e __v850e__ __v850e2 __v850e2__	Decimal constant 1.	__RH850 __RH850__ __v850e3v5 __v850e3v5__ (note2)
__CA850 __CA850__	Decimal constant 1.	__CCRH __CCRH__ (note2)
__CHAR_SIGNED__	Decimal constant 1. (Defined when signed is specified by the -Xchar option or when the -Xchar option is not specified).	__CHAR_SIGNED__ (note2)
__CHAR_UNSIGNED__	Decimal constant 1 (Defined when unsigned is specified by the -Xchar option)	None.
__DOUBLE_IS_32BITS__ _DOUBLE_IS_32BITS	Decimal constant 1.	__DOUBLE_IS_32BITS__ (note2) (note3)
__CPUmacro__	Macro indicating the target CPU. Decimal constant 1. A character string indicated by "product type specification" in the device file with "_ _" prefixed and "_" or "_ _"suffixed is defined.	None. (note4)
__reg32__	Decimal constant 1. (Defined when the -reg=32 option is specified)	__reg32__ (note2) (note5)
__reg26__	Decimal constant 1. (Defined when the -reg=26 option is specified)	None.
__reg22__	Decimal constant 1. (Defined when the -reg=22 option is specified)	__reg22__ (note2) (note6)

Notes:

*1: This option is defined when the -Xansi option is specified.

*2: Values are not set.

*3: This macro is only valid when -Xdbl_size=4 is specified.

*4: The CC-RH compiler does not support device files.

*5: This macro is defined when the -Xreg_mode=32 option is specified.

*6: This macro is defined when the -Xreg_mode=22 option is specified.

Chapter 4 Extended Language Specifications

This chapter gives a comparison table listing correspondences between extended language specifications for the CC-RH and CA850 compilers.

Description	CA850	CC-RH
Description with assembler instruction (note1)	#pragma asm *assembler instruction* #pragma endasm	#pragma inline_asm *function-name [, function-name ...]*
	__asm(“*assembler instruction*”);	None.
Inline expansion specification	#pragma inline *function-name* *[, function-name ...]*	#pragma inline *function-name [, function-name ...]*
Data memory allocation (note2)	#pragma section *section-type* begin *Variable declarations and definitions* #pragma section *section-type* end	#pragma section *attribute strings* Variable declarations and definitions #pragma section default
	#pragma section *section-type* *“ section-name”* begin *Variable declarations and definitions* #pragma section *section-type* *“section-name”* end	#pragma section *attribute strings ["section name"]* Variable declarations and definitions #pragma section default
Program memory allocation	#pragma text *“section-name”* *function-name*	#pragma section text *["section name"]* Variable definitions #pragma section default
	pragma text *“ section-name”*	
Peripheral I/O register name validation specification (note3)	#pragma ioreg	None.
Interrupt/exception handler specification (note4)	#pragma interrupt interrupt-request-name *function-name* *allocation-method* __interrupt function-definition, or function-declaration	#pragma interrupt *function-name* *[interrupt specification]*
	#pragma interrupt interrupt-request-name *function-name* *allocation-method* __multi_interrupt function-definition, or function-declaration	#pragma interrupt *function-name* (enable=true, *[interrupt specification]*)
Interrupt disable function specification	#pragma block_interrupt *function-name*	#pragma block_interrupt *function-name*
Task specification	#pragma rtos_task *function-name*	None.
Structure type packing specification	#pragma pack ([1|2|4|8])	#pragma pack ([1|2|4])

Notes:

*1: CA850 uses an extended statement when assembler instructions are embedded in functions written in the C language. CC-RH treats the function itself as an assembler instruction and inline expands assembly-language functions declared with #pragma inline_asm at the sites of function calls.

*2: The specifiable types and attribute strings of sections differ between the CA850 and CC-RH compilers.

*3: CC-RH does not support device files; access is by including “iodefine.h”, the header file for peripheral I/O registers.

*4: With the CA850 compiler, branch instructions for functions written as interrupt functions are automatically allocated to the interrupt handler addresses. With CC-RH, the user must define and allocate interrupt and exception vectors.

Chapter 5 Assembler Directives

This chapter gives a comparison table listing correspondences between assembler directives for the CC-RH and CA850 compilers. Directives are handled directly by the assembler executing a sequence of processing and do not become machine code.

Description	CA850	CC-RH
Section Definition Quasi Directives	.bss	.dseg bss
	.const	.cseg const
	.data	.dseg data
	.previous	None.
	.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	None.
	.vdebug	None.
	.vline	None.
Symbol Control Quasi Directives	.ext_ent_size	None. (note1)
	.ext_func	None.
	.file	.file
	.frame	None.
	.set	.set
	.size	None.

Description	CA850	CC-RH
Location Counter Control Quasi Directives	.align	.align
	.org	.offset (note2)
Area Allocation Quasi Directives	.byte	.db
	.hword	.db2 .dhw
	.shword	.dshw
	.word	.db4 .dw
	.float	.float
	.lcomm	None.
	.space	.ds
	.str	.db
Program Linkage Quasi Directives	.globl	.public
	.extern	.extern
	.comm	None.
Assembler Control Quasi Directive	.option reg_mod	$REG_MODE
	.option nomacro	$NOMACRO
	.option macro	$MACRO
	.option data	$DATA
	.option sdata	$SDATA
	.option nowarning	$NOWARNING
	.option warning	$WARNING
File Input Control Quasi Directives	.binclude	$binclude
	.include	$include
Repetitive Assembly Quasi Directives	.irepeat	.irp
	.repeat	.rept

Description	CA850	CC-RH
Conditional Assembly Quasi Directives	.else	$else
	.elseif	$elseif
	.elseifn	$elseifn
	.endif	$endif
	.if	$if
	.ifdef	$ifdef
	.ifn	$ifn
	.ifndef	$ifndef
Skip Quasi Directives	.exitm	.exitm
	exitma	.exitma
Macro Quasi Directives	.macro	.macro
	.local	.local
	.endm	.endm

Notes:

*1: No relinking function is provided.

*2: In the CC-RH assembler, “.org” is a directive that indicates the start of an absolute address section.

Chapter 6 Peripheral I/O Registers

This chapter describes the handling of peripheral I/O registers by the CA850 and CC-RH compilers.

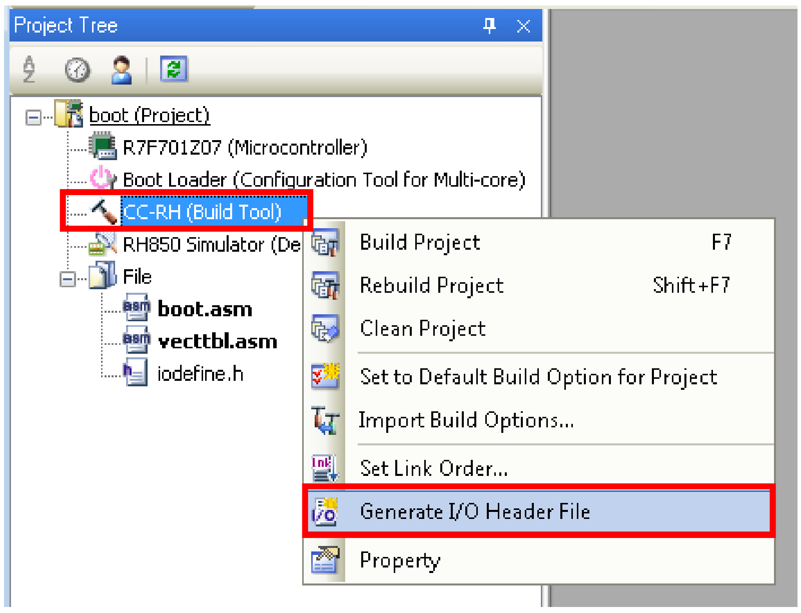
6.1 Peripheral I/O Registers in CA850

For the CA850 compiler, when the #pragma ioreg directive is added, access to peripheral I/O registers in the C language is possible by using the names of the registers. These names and the corresponding addresses are registered in a device file and the names are converted into addresses during assembly. The names of registers that are registered in a device file are described in the user's manual for each MCU.

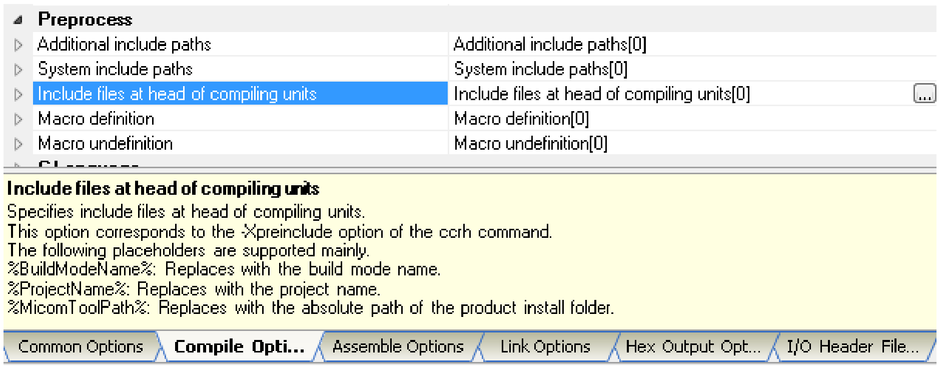
6.2 Peripheral I/O Registers in CC-RH

Rather than having device files, the CC-RH compiler supports access to peripheral I/O registers through the inclusion of "iodefine.h", a list of the names of peripheral I/O registers and the corresponding addresses.

When a new project is created in the CS+, the CS+ generates an I/O header file "iodefine.h" for the target MCU specified in the project and registers it as a source file in the project. The I/O header file defines the names of the registers provided in the MCU and their addresses. The header file can also be generated by right-clicking the [CC-RH (Build Tool)] node in the CS+ project tree and then clicking [Generate I/O Header File].



When accessing a register in a C-language program, include the I/O header file. By specifying the header file as a parameter for the –Xpreinclude option, the #include specification can be omitted from the source file. The –Xpreinclude option can be specified by selecting the [Compile Options] tab => [Preprocess] category => [Include files at head of compiling units]. In this property setting, specify the I/O header file for the target MCU.



Preprocess	
Additional include paths	Additional include paths[0]
System include paths	System include paths[0]
Include files at head of compiling units	Include files at head of compiling units[0]
Macro definition	Macro definition[0]
Macro undefinition	Macro undefinition[0]

Include files at head of compiling units
Specifies include files at head of compiling units.
This option corresponds to the -Xpreinclude option of the ccrh command.
The following placeholders are supported mainly.
%BuildModeName%: Replaces with the build mode name.
%ProjectName%: Replaces with the project name.
%MicomToolPath%: Replaces with the absolute path of the product install folder.

Chapter 7 Interrupts and Exceptions

This chapter describes interrupt and exception handlers for the CA850 and CC-RH compilers.

7.1 Interrupts and Exceptions in CA850

The CA850 compiler places branch instructions for functions specified with ‘function-name’ at the handler address corresponding to ‘interrupt-request-name’ specified by the #pragma interrupt directive. Specify the name of an interrupt request registered in the device file as ‘interrupt-request-name’. The details of ‘interrupt-request-name’ are described in the user’s manual for each MCU.

```
#pragma interrupt interrupt-request-name function-name allocation-method
```

When the __interrupt qualifier is added, the function is compiled as an interrupt function.

```
__interrupt function-definition, or function-declaration
```

When the __multi_interrupt qualifier is added, the function is compiled as a function for multiple interrupts.

```
__multi_interrupt function-definition, or function-declaration
```

For example, the handler address for the interrupt request with the name “INTP0” for the V850ES/FJ3 is 0xA0. In this case, using the following #pragma interrupt directive places a jr_funcint instruction from 0xA0. In addition, the #pragma interrupt directive compiles the intfunc function as the interrupt function and outputs processing for saving and restoring registers around the interrupt and exception handlers.

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

7.2 Interrupts and Exceptions in CC-RH

When a #pragma interrupt directive is specified, the CC-RH compiles the function specified in "function name" according to the specification in "interrupt specification".

```
#pragma interrupt function-name [interrupt specification]
```

For example, the "func" function is compiled as an interrupt function according to the #pragma interrupt directive shown below. In addition, the processing for saving and restoring the ctpc, ctpsw, fpepc, and fpsr and the ei and di instructions are output according to the interrupt specifications.

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

Note that the user should define and allocate interrupt and exception vectors in the CC-RH. When a new project file is created in the CS+, the "vecttbl.asm" file is registered as a source file and it defines the format for interrupt/exception vectors. Customize the file as necessary and allocate vectors to appropriate addresses in accordance with the target MCU. The following describes the interrupt/exception vectors in "vecttbl.asm".

a. RESET

The following definition embeds the "jr32 __start" instruction at the head of the RESET section.

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

For example, when a new project is created for the RH850/F1L by the CS+, the "-start" linkage editor option specifies the allocation of the RESET section at address 0x00   that is, the "jr32 __start" instruction is embedded at address 0x00.

b. Interrupts and exceptions in direct vector method

The base location for handler addresses is obtained by adding the base address indicated by the RBASE or EBASE register and the offset specific to the exception source. Either the RBASE or EBASE register is selected through the PSW.EBV bit. The following definition assumes RBASE as the base address and allocates interrupt/exception handlers immediately after RESET.

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

            .align      16
            jr32        _Dummy ; SYSERR

            .align      16
            jr32        _Dummy ; HVTRAP
            ...
```

In the "vecttbl.asm" file, an instruction for branching to the dummy function "_Dummy" is specified at the offset locations corresponding to SYSERR, HVTRAP, FETRAP, etc. The "_Dummy" function is a routine that repeats branches to itself. Customize it as necessary.

Modify "_Dummy" to "_interrupt-function name" at the offset locations corresponding to the exceptions and interrupts that should be customized. In addition, define the interrupt functions through the #pragma interrupt directive. The following shows an example for executing the interrupt function "func" when an exception "SYSERR" occurs.

```
        .section "RESET", text
        .align    512
        jr32      __start ; RESET
                             Modify "_Dummy" to "_interrupt-function name".
        .align    16
        jr32      _func ; SYSERR

        .align    16
        jr32      _Dummy ; HVTRAP
        ...
```

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

c. Interrupts and exceptions in table lookup method

Interrupts can be specified in the table lookup method, which is an extended specification for interrupts. In the direct vector method, only one handler address is assigned to each priority level of EI-level interrupts; for all interrupt channels having the same priority level, execution therefore branches to the same interrupt handler address. However, there will be cases where the application requires a separate code area to be used for each interrupt handler. To implement this, the CC-RH provides the table lookup method.

```
        .section "EIINTTBL", const
        .align    512
        .dw       #_Dummy_EI ; INT0
        .dw       #_Dummy_EI ; INT1
        .dw       #_Dummy_EI ; INT2
        .rept     288 - 3
        .dw       #_Dummy_EI ; INTn
        .endm
```

In the "vecttbl.asm" file, an interrupt/exception table for the table lookup method is defined in the EIINTTBL section. When a new project file is created for the RH850/F1L by the CS+, the "-start" linkage editor option specifies allocation of the table immediately after the RESET section.

The addresses where the dummy function "_Dummy_EI" is stored are specified in areas offset from the head of the EIINTTBL section by an address of a multiple of four. Thus, execution branches to _Dummy_EI when an exception/interrupt at interrupt priority level n in the table lookup method occurs. The "_Dummy_EI" function is a routine that repeats branches to itself. Customize it as necessary.

Modify "#_Dummy_EI" to "#_interrupt-function name" at the offset locations corresponding to the channels that should be customized. In addition, define the interrupt functions through the #pragma interrupt directive. The following shows an example for executing the interrupt function "func" when a channel-9 interrupt "EIINT9" occurs.

```
.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
```

Modify "#_Dummy_EI" to "*#_interrupt-function name*".

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

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

Note that the direct vector method is the default exception/interrupt method in the RH850; to switch to the table lookup method, modify the interrupt control register value.

Chapter 8 ROMization

The data for variables with initial values should be stored in ROM and then copied to RAM before such variables are accessed after the MCU is reset. This sequence is called ROMization. The ROMization processing differs between the CA850 and CC-RH. This section describes ROMization processing in the CX and CC-RH.

8.1 ROMization Processing in CA850

For the CA850 compiler, a section (.sdata or .data) to which variables having initial values are allocated will be ROMized by default. The initial-value data are allocated to addresses in ROM to which ROMization area reservation code (rompcrt.o) has been allocated and copied from ROM to RAM using the _rcopy function. Addresses for the copy destination (addresses for the .sdata or .data section) are specified by a link directive file (*.dir).

The following shows an example of C source code calling the copy function when rompcrt.o is used for ROMization.

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

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

The symbol '_S_romp' is defined in the ROMization area reservation code file (rompcrt.o). '&_S_romp' is the address where the initial-value data which have been stored in ROM start and is automatically determined by the linker.

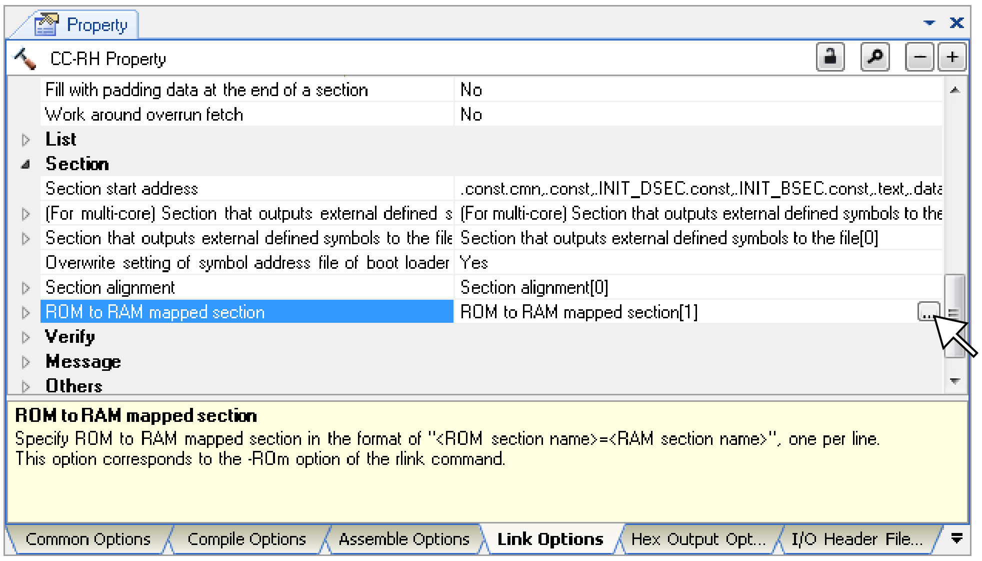
8.2 ROMization Processing in CC-RH

a. Specifying ROMization

In the CC-RH, the target sections for ROMization should be specified through the "–rom" linkage editor option. <ROM-section name> is a target section for ROMization. Use the "–start" linkage editor option to allocate the sections specified as <ROM-section name> to ROM and those specified as <RAM-section name> to RAM.

```
-rom=<ROM-section name>=<RAM-section name>
```

In the CS+, select the [Link Options] tab => [Section] category, click the [...] button at the right end of the [ROM to RAM mapped section] row, and specify the sections to be copied from ROM to RAM in the format of <ROM-section name>=<RAM-section name> with one section per line.



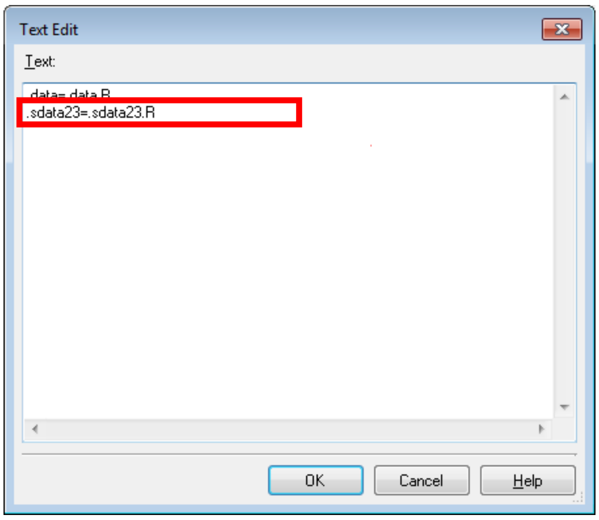
Fill with padding data at the end of a section	No
Work around overrun fetch	No
Section start address	.const.cmn,.const,.INIT_DSEC.const,.INIT_BSEC.const,.text,.data
[For multi-core] Section that outputs external defined s	[For multi-core] Section that outputs external defined symbols to the
Section that outputs external defined symbols to the file	Section that outputs external defined symbols to the file[0]
Overwrite setting of symbol address file of boot loader	Yes
Section alignment	Section alignment[0]
ROM to RAM mapped section	ROM to RAM mapped section[1]

ROM to RAM mapped section
Specify ROM to RAM mapped section in the format of "<ROM section name>=<RAM section name>", one per line.
This option corresponds to the -ROm option of the rlink command.

When a new project is created by the CS+, the following options are specified by default.

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

When an additional section other than the .data section is specified to store variables with initial values, use the "–rom" option to add this section to the target sections for ROMization. The following shows an example when the .sdata23 section is added as a target of ROMization (the RAM section name is .sdata.R).



```
.data=.data.R
.sdata23=.sdata23.R
```

b. Defining the initialization table

In the CC-RH, the ROMized data should be copied from ROM to RAM by using the "_INITSCT_RH" function. When a new project file is created by the CS+, the startup routine "cstartm.asm" is registered as a source file and it defines the initialization table to be used to copy the data of variables with initial values as follows.

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

The initialization table is allocated to the .INIT_DSEC.const section, and a 4-byte area is allocated to each of the .data section start address, .data section end address, and .data.R section start address in that order.

Prefixing a section name with "__s" generates a reserved symbol that has the start address of the section as its value. Likewise, prefixing a section name with "__e" generates a reserved symbol that has the end address of the section as its value. Using these reserved symbols is recommended for additional specifications to the initialization table.

When an additional section is specified to store variables with initial values, add the start and end addresses of the section in this initialization table.

```
;-----------------------------------------------------------------------------
;               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
```

In the CC-RH, the "_INITSCT_RH" function can also be used to initialize with zero the sections where variables without initial values are to be stored. The startup routine "cstartm.asm" defines the zero-initialization table.

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

The zero-initialization table is allocated to the .INIT_BSEC.const section, and a 4-byte area is allocated to each of the .bss section start address and .bss section end address in that order. When an additional section other than the .bss section is specified to store variables without initial values, add the addresses of the section in the same format as the existing settings.

c. Calling the copy function

The "_INITSCT_RH" function is called from the startup routine "cstartm.asm". This processing initializes the sections defined in each table.

```
        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
```

Chapter 9 Section Allocation

This chapter describes how to allocate sections with the CA850 and CC-RH compilers.

9.1 Section Allocation in CA850

With the CA850 compiler, addresses for the allocation of sections are stated in a link directive file (*.dir). When a link directive file is input to the linker, the addresses for the allocation of sections are specified. The following shows the format of link directive files for the CA850 compiler.

```
Segment name: !segment type   ?segment attribute   Vaddress {

        Output-section name=$section type   ?section attribute   input-section name;
        Output-section name=$section type   ?section attribute   input-section name;
        …
} ;
```

As shown below, the .const section is allocated to 0x1000 and .pro_epi_runtime and .text sections in order are allocated to rising address ranges from the end of the .const section. After that, .data, .sdata, .sbss, and .bss sections are allocated to rising addresses in order from 0xfedf6000.

```
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 ;
};
```

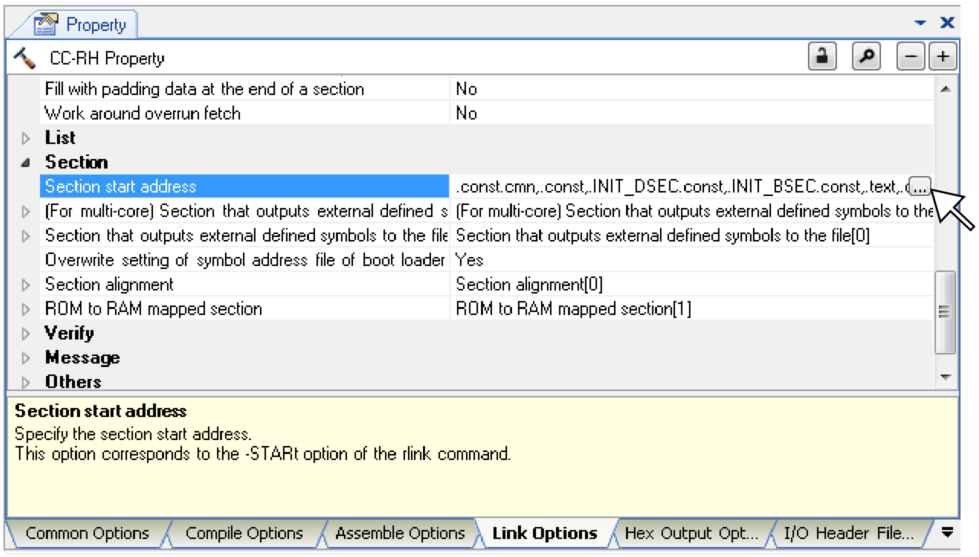
9.2 Section Allocation in CC-RH

In the CC-RH, section allocation addresses should be specified through the "-start" linkage editor option; the link directive file in the CX or the like is not used. The following shows an example for specifying the "–start" option in the CC-RH. For details, refer to the user's manual for the build process.

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

Through the above option settings, allocation of the RESET section begins from address 0x00. Allocation of the EIINTTBL, .const, .INIT_DSEC.const, .INIT_BSEC.const, .text, and .data sections begins after the end of the RESET section and proceeds toward higher addresses in that order. Allocation of the .data.R, .bss, and .stack.bss sections begins from address 0xFEDE0000 and proceeds toward higher addresses in that order.

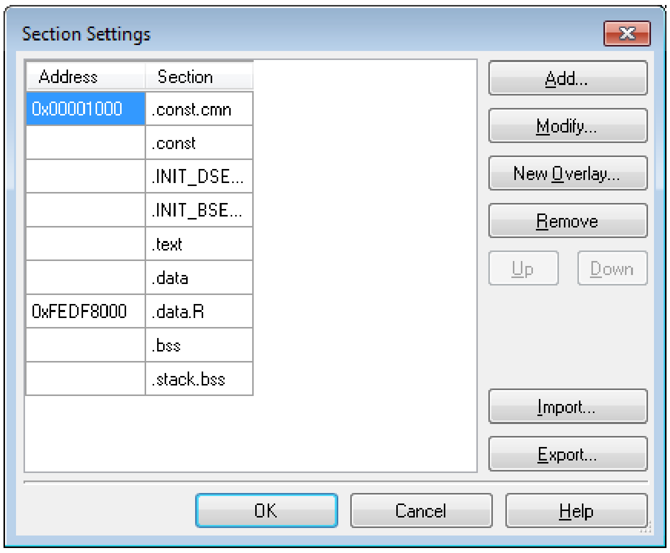
In the CS+, section allocation can be specified through the GUI; select the [Link Options] tab => [Section] category, and click the [...] button at the right end of the [Section start address] row.



Fill with padding data at the end of a section	No
Work around overrun fetch	No
Section start address	.const.cmn,.const,.INIT_DSEC.const,.INIT_BSEC.const,.text,.d
[For multi-core] Section that outputs external defined s	[For multi-core] Section that outputs external defined symbols to the
Section that outputs external defined symbols to the file	Section that outputs external defined symbols to the file[0]
Overwrite setting of symbol address file of boot loader	Yes
Section alignment	Section alignment[0]
ROM to RAM mapped section	ROM to RAM mapped section[1]

Section start address
Specify the section start address.
This option corresponds to the -STARt option of the rlink command.

The Section Settings dialog box will open; addresses and sections can be added and modified through manipulation in this dialog box.



Address	Section
0x00001000	.const.cmn
	.const
	.INIT_DSE...
	.INIT_BSE...
	.text
	.data
0xFEDF8000	.data.R
	.bss
	.stack.bss