Section 1 Preface

This document describes the following points that should be especially borne in mind for migration from the SuperH family C/C++ compiler (hereafter called the SHC) to the RH850 family C compiler (hereafter called the CC-RH).

- Options
- Intrinsic functions
- Extended language specifications
- Program compatibility
- Assembler directives and macro functions
- Others

Notes:
In the CC-RH,
- The startup routine and IO header file for the SHC cannot be used.
- The Standard Library Generator for SHC is not included.
- The DSP-C Language Specifications and C++ Language Specifications are not supported.

The following shows the version of compilers that this document is targeted.

Compiler	Version
CC-RH	V1.03.00
SHC	V9.04 Release03

Section 2 Options

This section shows the correspondence between the SHC and CC-RH options.
Note that the SHC does not distinguish between uppercase and lowercase letters, but the CC-RH distinguishes between them in the compiler options and assembler options. However, it does not distinguish between them in the linkage editor options.

2.1 Compiler Options

2.1.1 Source Options

No	Description	SHC	CC-RH
1	Displays the descriptions of options	Execute SHC command on the command line.	-h
2	Specifies the include-file include path name	-include	-I
3	Includes the specified files at the head of compiling units	-preinclude	-Xpreinclude
4	Defines macro names	-define	-D
5	Information message	-message -nomessage	-
6	Specifies the path name from which a file for inter-file inline expansion is to be obtained.	-file_inline_path	-
7	Changes the message level.	-change_message	-

2.1.2 Object Options

No	Description	SHC	CC-RH
1	Outputs the source program after preprocessor expansion	-preprocessor -noline	-P -Xpreprocess #line is not output by default (same as the –noline option of the SHC). When -Xpreprocess=line is specified, #line is output.
2	Object type	-code=machinecode -code=asmcode	-c -S
3	Debugging information	-debug -nodebug	-g
4	Section name	-section	-Xsection
5	Area of string literals to be output	-string	None String literals are always assigned to the section having the const attribute.
6	Specifies the object file name	-objectfile	-o
7	Generates template instances	-template	None
8	Specifies the memory space where the label addresses or runtime routines belonging to the specified section are to be allocated	-abs16/20/28/32	None
9	Method of division	-division=cpu	None
10	Disables saving and restoring of floating-point registers	-ifunc	None
11	16-byte or 32-byte alignment of labels	-align16 -align32 -noalign	None
12	TBR relative function call	-tbr	None
13	Order of uninitialized variables	-bss_order	None
14	Disposes variables	-stuff	None

15	Disposes variables in $G0/$G1	-stuff_gbr	None
16	Aligns branch destinations	-align4	-Xalign4
17	Allocates const volatile variables	-const_volatile	const volatile variables are always assigned to the section having the const attribute.

2.1.3 List Options

No	Description	SHC	CC-RH
1	Outputs the list file	-listfile	None Information equivalent to the compiler list file for the SHC is output in the assemble list file. Refer to the description of -Xasm_option=-Xprn_path.
2	List contents and format	-show	None The location counter values, codes, line numbers, source program contents, and type and size of sections are output in the assemble list file.

2.1.4 Optimize Options

No	Description	SHC	CC-RH
1	Optimization level	-optimize	Refer to 2.1.5 Details of Optimize Options.
2	Optimization with the execution speed precedence or the code size precedence	-speed -size -nospeed	Refer to 2.1.5 Details of Optimize Options.
3	Inter-module optimization information	-goptimize	None
4	Optimized for access to external variables	-map	-Omap
5	Optimization of external variable access	-smap	-Osmap
6	Automatically creates GBR relative access codes	-gbr	None
7	switch statement expansion method	-case	-Xswitch
8	Shift-operation expansion	-shift	None
9	Transfer-code expansion	-blockcopy	None
10	Unaligned data transfer	-unaligned	None
11	Automatic inline expansion	-inline -noinline	-Oinline
12	Inter-file inline expansion	-file_inline	None
13	Handles external variables as volatile	-global_volatile	-Xvolatile
14	External variable optimizing range	-opt_range	None
15	Eliminates vacant loops	-del_vacant_loop	None
16	Loop unrolling	-loop -noloop	-Ounroll
17	Maximum number of loop expansions	-max_unroll	-Ounroll
18	Eliminates expressions preceding infinite loops	-infinite_loop	None
19	Allocates external variables to registers	-global_alloc	None
20	Allocates structure/union members to registers	-struct_alloc	None
21	Propagates const constant	-const_var_propagate	None
22	Expands constant loading instructions	-const_load	None
23	Instruction scheduling	-schedule	-Opipeline
24	Software pipelining	-softpipe	None
25	Divides optimizing ranges	-scope	None
26	Generates GBR relative logic operation	-logic_gbr	None
27	Prevents expansion of C++ Inline functions	-cpp_noinline	None
28	Optimization considering the type of the object indicated by the pointer	-alias	-Xalias

2.1.5 Details of Optimize Options

The SHC optimize options differ from the CC-RH options in their default values and format; the user should modify their specifications. Note also that the optimization functions are not the same between the SHC and CC-RH.
The following shows some combinations of SHC optimize options and their corresponding CC-RH options.

No	SHC		CC-RH
1	-optimize=debug_only	-size	-Onothing
		-speed	
		-nospeed	
2	-optimize=0	-size	-Odefault
		-speed	
		-nospeed	
3	-optimize=1	-size	-Osize
		-speed	-Ospeed
		-nospeed	-Ospeed,-Ounroll=1,-Oinline=1

2.1.6 Other Options

No	Description	SHC	CC-RH
1	Checks the syntax according to the Embedded C++ Language Specifications	-ecpp	None
2	Checks the syntax according to the DSP-C Language Specifications	-dspc	None
3	Comment nesting (/**/)	-comment	None
4	Guarantees the MAC register contents	-macsave	None
5	Saves and restores SSR and SPC	-save_cont_reg	None
6	Return value extension	-rtnext	None
7	Converts the floating-point constant divisions to multiplications	-approxdiv	None
8	Avoids SH7055 illegal operation	-patch=7055	None
9	FPSCR register switching	-fpscr	None
10	Suppresses optimization of loop iteration conditions	-volatile_loop	None
11	Automatically selects the enumeration data size	-auto_enum	-Xenum_type=auto
12	Preferential allocation of register storage class variables	-enable_register	None
13	Conforms to the ANSI standard for the following processing: – unsigned int and long type operations – Associativity of floating-point operations	-strict_ansi	None
14	Converts integer divisions to floating-point divisions	-fdiv	None
15	Handles floating-point constants as fixed-point constants	-fixed_const	None
16	Handles 1.0r (1.0R) as the maximum value of _ _fixed (long _ _fixed) type	-fixed_max	None
17	Omits type conversion for _ _fixed multiplication result	-fixed_noround	None
18	DSP repeat loop	-repeat	None
19	Omits the range check for conversion between floating-point number and integer	-simple_float_conv	None
20	Suppresses DIVS and DIVU instruction generation	-nouse_div_inst	None
21	Changes the operation order for floating-point expressions	-float_order	None

2.1.7 CPU Options

No	Description	SHC	CC-RH
1	CPU/operating mode	-cpu	-Xcpu -Xcommon
2	Byte order	-endian	None
3	Floating-point operation mode	-fpu = { single | double } -double=float	-Xdbl_size Note: This option specifies the data size of double and long double type
4	Rounding mode	-round	-Xround
5	Denormalized numbers	-denormalize	None
6	Program section position independent	-pic	None
7	Specifies the bit field order	-bit_order	-Xbit_order
8	Boundary alignment of structure, union, and class members	-pack	-Xpack
9	Exception handling	-exception	None
10	Runtime type information	-rtti	None
11	Method of division	-division	None

2.1.8 Options Other than Above

No	Description	SHC	CC-RH
1	Selects C or C++ language	-lang	None
2	Disables copyright output	-logo -nologo	None
3	Selects the character code in string literals	-euc -sjis -latin1	-Xcharacter_set
4	Specifies the Japanese character code within the object	-outcode	None
5	Specifies the subcommand file	-subcommand	@

2.2 Assembler Options

2.2.1 Source Options

No	Description	SHC	CC-RH
1	Displays the descriptions of options	Execute the asmsh command on the command line.	-h
2	Specifies the include-file include path name	-include	-I
3	Defines replacement symbols	-define	-D
4	Defines integer preprocessor variables	-assigna	None
5	Defines character preprocessor variables	-assignc	None

2.2.2 Object Options

No	Description	SHC	CC-RH
1	Debugging information	-debug	-g
2	Preprocessor expansion result	-expand	None
3	Literal pool output point	-literal	None
4	Controls object module output	-object	-o
5	Specifies the size of unresolved symbols	-dispsize	None

2.2.3 List Options

No	Description	SHC	CC-RH
1	Controls output of the assemble listing	-list	-Xprn_path
2	Controls output of the source program listing	-source	None
3	Controls output of parts of source program listing and sets the size of tabs	-show	None
4	Controls output of the cross-reference listing	-cross_reference	None
5	Controls output of the section information listing	-section	None

2.2.4 Other Options

No	Description	SHC	CC-RH
1	Specifies the size mode for automatic literal pool generation	-auto_literal	None
2	Prevents output of information on unreferenced external symbols	-exclude	None
3	Specifies to check privileged-mode instructions	-chkmd	None
4	Specifies to check LDTLB instructions	-chktlb	None
5	Specifies to check cache-related instructions	-chkcache	None
6	Specifies to check DSP-related instructions	-chkdsp	None
7	Specifies to check FPU-related instructions	-chkfpu	None
8	Specifies to check 8-byte boundary alignment of .FDATA	-chkalign8	None

2.2.5 CPU Options

No	Description	SHC	CC-RH
1	Specifies the target CPU	-cpu	-Xcpu -Xcommon
2	Specifies the endian type	-endian	None
3	Specifies the rounding mode for floating-point data	-round	None
4	Specifies handling of denormalized numbers in floating-point data	-denormalize	None

2.2.6 Options Other than Above

No	Description	SHC	CC-RH
1	Changes the error level at which the assembler is abnormally terminated	-abort	None
2	Specifies the character code in the source file	-latin1 -sjis -euc	-Xcharacter_set
3	Specifies the Japanese character for the output to the object code.	-outcode	None
4	Sets the number of lines in the assemble listing	-lines	None
5	Sets the number of digits in the assemble listing	-columns	None
6	Copyright	-logo -nologo	None
7	Specifies the subcommand file	-subcommand	@

2.3 Optimizing Linkage Editor Options

In the CC-RH, linkage editor options can be specified in the abbreviated form in the same way as in the SHC. The uppercase letters in the options and parameters in the following tables indicate abbreviated forms.

2.3.1 Input Options

No	Description	SHC	CC-RH
1	Specifies the input file	-Input	-Input
2	Specifies the input library file	-LIBrary	-LIBrary
3	Specifies the input binary file	-Binary	-Binary
4	Defines undefined symbols	-DEFine	-DEFine
5	Specifies the execution start address	-ENTry	-ENTry
6	Disables prelinker initiation	-NOPRElink	None

2.3.2 Output Options

No	Description	SHC	CC-RH
1	Output format	-FOrm	-FOrm
2	Debugging information	-DEBug	-DEBug
3	Unifies the record size	-REcord	-RECord
4	ROM support function	-ROm	-ROm
5	Specifies the output file	-OUtput	-OUtput
6	External symbol allocation information file	-MAp	-MAp
7	Output to unused areas	-SPace	-SPace
8	Information message	-Message -NOMessage	-Message -NOMessage
9	Notifies the user of unreferenced defined symbols	-MSg_unused	-MSg_unused
10	Reduces empty areas of boundary alignment	-DAta_stuff	None
11	Specifies the data record byte count	-BYte_count	-Byte_count
12	Calculates the cyclic redundancy check (CRC)	-CRc	-CRc
13	Outputs padding data at the end of a section	- PADDING	-PADDING

2.3.3 List Options

No	Description	SHC	CC-RH
1	Outputs the list file	-LISt	-LISt
2	List contents	-SHow	-SHow

2.3.4 Optimize Options

No	Description	SHC	CC-RH
1	Optimization	-OPtimize	None
2	Specifies the minimum size for same-code unification	-SAMESize	None
3	Profile information file	-PROfile	None
4	Cache size	-CAchesize	None
5	Partially disables optimization	-SYmbol_forbrid -SAMECode_forbrid -Variable_forbrid -FUnction_forbrid -SEction_forbrid -Absolute_forbrid	None

2.3.5 Section Options

No	Description	SHC	CC-RH
1	Specifies a section start address	-STARt	-STARt
2	Outputs externally defined symbol addresses to a definition file	-FSymbol	-FSymbol
3	Changes the section alignment value to 16 bytes	-ALIGNED_SECTION	-ALIGNED_SECTION

2.3.6 Verify Options

No	Description	SHC	CC-RH
1	Checks addresses	-CPu	-CPu
2	Checks for physical space overlapping	-PS_check	None
3	Stops dividing the specified section	-CONTIGUOUS_SECTION	None

2.3.7 Other Options

No	Description	SHC	CC-RH
1	Always outputs the S9 record	-S9	-S9
2	Stack information file	-STACk	-STACk
3	Compresses debugging information	-COmpress -NOCOmpress	-COmpress -NOCOmpress
4	Reduces memory occupancy	-MEMory	-MEMory
5	Modifies symbol names	-REName	-REName
6	Deletes symbol names	-DELete	-DELete
7	Replaces modules	-REPlace	-REPlace
8	Extracts modules	-EXTract	-EXTract
9	Deletes debugging information	-STRip	-STRip
10	Changes message levels	-CHange_message	-CHange_message
11	Hides local symbol names	-Hide	-Hide
12	Shows the total sizes of sections	-Total_size	-Total_size
13	Information file for the emulator	-RTs_file	None

2.3.8 Subcommand File Options

No	Description	SHC	CC-RH
1	Subcommand file	-SUbcommand	-SUbcommand

2.3.9 CPU Options

No	Description	SHC	CC-RH
1	Specifies the SBR address	-SBr	None

2.3.10 Options Other Than Above

No	Description	SHC	CC-RH
1	Copyright	-LOgo -NOLOgo	-Logo -NOLOgo
2	Continuation	-END	-END
3	Termination	-EXIt	-EXIt

Section 3 Intrinsic Functions

If an SHC intrinsic function is used for the CC-RH, the following compilation error message will be output.

E0562310: Undefined external symbol "symbol" referenced in "file"

If <machine.h>, <smachine.h>, or <umachine.h> is included, the following compilation error message will be output. In the CC-RH, no header file needs to be included to use intrinsic functions.

F0520005: Could not open source file "file".

The following shows the CC-RH intrinsic functions that execute the processes equivalent to those of the SHC intrinsic functions.

No	Description	SHC	CC-RH
1	Writes to the status register (SR)	void set_cr(int cr)	None
2	Reads the status register (SR)	int get_cr(void)	None
3	Writes to the interrupt mask bit	void set_imask(int mask)	void __set_il_rh(int NUM, void* ADDR);
4	Reads the interrupt mask bit	int get_imask(void)	Reads PMR by using the following code; unsigned long __stsr_rh(long regID, long selID); __stsr_rh(11, 2)
5	Writes to the vector base register (VBR)	void set_vbr(void *base)	void __ldsr_rh(long regID, long selID, unsigned long a);(Note 1)
6	Reads the vector base register (VBR)	void *get_vbr(void)	unsigned long __stsr_rh(long regID, long selID);(Note 1)
7	Writes to the global base register (GBR)	void set_gbr(void *base)	None(Note 2)
8	Reads the global base register (GBR)	void *get_gbr(void)	None
9	Reads a GBR-based byte	unsigned char gbr_read_byte(int offset)	None
10	Reads a GBR-based word	unsigned short gbr_read_word(int offset)	None
11	Reads a GBR-based longword	unsigned long gbr_read_long(int offset)	None
12	Writes to a GBR-based byte	void gbr_write_byte(int offset, unsigned char data)	None
13	Writes to a GBR-based word	void gbr_write_word(int offset, unsigned short data)	None
14	Writes to a GBR-based longword	void gbr_write_long (int offset, unsigned long data)	None
15	ANDs a GBR-based byte	void gbr_and_byte(int offset, unsigned char mask)	None
16	ORs a GBR-based byte	void gbr_or_byte(int offset, unsigned char mask)	None
17	XORs a GBR-based byte	void gbr_xor_byte(int offset, unsigned char mask)	None
18	Tests a GBR-based byte	void gbr_tst_byte(int offset, unsigned char mask)	None
19	SLEEP instruction	void sleep(void)	void __halt()
20	TAS instruction	int tas(char *addr)	None
21	TRAPA instruction	int trapa (int trap_no)	None(Note 3)
22	OS system call	int trapa_svc (int trap_no, int code, type1 para1, type2 para2, type3 para3, type4 para4)	None(Note 4)

Note 1: The VBR in the SH corresponds to the RBASE, EBASE, and INTBP in the RH850.
Note 2: The GBR in the SH corresponds to the GP and EP in the RH850. For the CC-RH, specify the GP and EP in assembly language.
Note 3: The TRAPA instruction in the SH corresponds to the TRAP instruction in the RH850.
Note 4: The OS system calls in the SH correspond to the SYSCALL instruction in the RH850.

23	PREF instruction	void prefetch(void *p)	None
24	TRACE instruction	void trace(long v)	None
25	LDTLB instruction	void ldtlb(void)	None
26	NOP instruction	void nop(void)	void __nop(void)
27	Upper 32 bits of the result of a signed 64-bit multiplication	long dmuls_h(long data1, long data2)	long __mul32(long data1, long data2)
28	Lower 32 bits of the result of a signed 64-bit multiplication	unsigned long dmuls_l(long data1, long data2)	None
29	Upper 32 bits of the result of an unsigned 64-bit multiplication	unsigned long dmulu_h(unsigned long data1,unsigned long data2)	unsigned long __mul32u(unsigned long data1, unsigned long data2)
30	Lower 32 bits of the result of an unsigned 64-bit multiplication	unsigned long dmulu_l(unsigned long data1,unsigned long data2)	None
31	SWAP.B instruction	unsigned short swapb(unsigned short data)	None (Note 5)
32	SWAP.W instruction	unsigned long swapw(unsigned long data)	long __hsw(long data)
33	Reverses the byte order inside 4-byte data	unsigned long end_cnvl(unsigned long data)	long __bsw(long data)
34	MAC.W instruction	int macw(short *ptr1, short *ptr2, unsigned int count) int macwl(short *ptr1, short *ptr2, unsigned int count, unsigned int mask)	None
35	MAC.L instruction	int macl(int *ptr1, int *ptr2, unsigned int count) int macll (int *ptr1, int *ptr2, unsigned int count, unsigned int mask)	None
36	Sets FPSCR	void set_fpscr(int cr)	Specifies x in FPSCR by using the following code; void __ldsr_rh(long regID, long selID, unsigned long a); __ldsr_rh(6, 0, x)
37	Refers to FPSCR	int get_fpscr(void)	None
38	FIPR instruction	float fipr(float vect1[4], float vect2[4])	None
39	FTRV instruction	void ftrv(float vec1[4], float vec2[4])	None
40	Transforms a 4-dimensional vector by a 4×4 matrix, and adds the result to a 4-dimensional vector	void ftrvadd(float vec1[4], float vec2[4], float vec3[4])	None
41	Transforms a 4-dimensional vector by a 4×4 matrix, and subtracts a 4-dimensional vector from the result	void ftrvsub(float vec1[4], float vec2[4], float vec3[4])	None
42	Performs addition of 4-dimension vectors	void add4(float vec1[4], float vec2[4], float vec3[4])	None
43	Performs subtraction of 4-dimension vectors	void sub4(float vec1[4], float vec2[4], float vec3[4])	None
44	Performs multiplication of 4×4 matrices	void mtrx4mul(float mat1[4][4], float mat2[4][4])	None
45	Performs multiplication and addition of 4×4 matrices	void mtrx4muladd(float mat1[4][4], float mat2[4][4], float mat3[4][4])	None
46	Performs multiplication and subtraction of 4×4 matrices	void mtrx4mulsub(float mat1[4][4], float mat2[4][4], float mat3[4][4])	None
47	Loads mat (4×4 matrix) to the extension register	void ld_ext(float mat[4][4])	None
48	Stores the extension register contents to mat (4×4 matrix)	void st_ext(float mat[4][4])	None
49	Computes the absolute value	long _ _fixed pabs_lf(long _ _fixed data) long _ _accum pabs_la(long _ _accum data)	None
50	Detects the MSB	_ _fixed pdmsb_lf(long _ _fixed data) _ _fixed pdmsb_la(long _ _accum data)	None

Note 5: This processing can be implemented by using the intrinsic function "long __bsh(long data)" in the CC-RH.

51	Shifts data arithmetically	long _ _fixed psha_lf(long _ _fixed data,int count) long _ _accum psha_la(long _ _accum data,int count)	None
52	Rounds data	_ _accum rndtoa(long _ _accum data) _ _fixed rndtof(long _ _fixed data)	None
53	Copies a bit pattern	long _ _fixed long_as_lfixed(long data) long lfixed_as_long(long _ _fixed data)	None
54	Specifies modulo addressing	void set_circ_x(_ _X _ _circ _ _fixed array[], size_t size) void set_circ_y(_ _Y _ _circ _ _fixed array[], size_t size)	None
55	Cancels modulo addressing	void clr_circ(void)	None
56	Specifies the CS bit value (DSR register)	void set_cs(unsigned int mode)	None
57	Computes the sine and cosine values	void fsca(long angle, float *sinv, float *cosv)	None
58	Computes the inverse of the square root	float fsrra(float data)	None
59	Invalidates the instruction cache block	void icbi(void *p)	None
60	Invalidates the cache block	void ocbi(void *p)	None
61	Purges the cache block	void ocbp(void *p)	None
62	Writes back the cache block	void ocbwb(void *p)	None
63	Prefetches instructions into the instruction cache	void prefi(void *p)	None
64	Synchronizes data operation	void synco(void)	None
65	Refers to the T bit	int movt(void)	None
66	Clears the T bit	void clrt(void)	None
67	Sets the T bit	void sett(void)	None
68	Extracts the middle 32 bits from contiguous 64 bits	unsigned long xtrct(unsigned long data1, unsigned long data2)	None
69	Adds two values and the T bit and sets the carry to the T bit	long addc(long data1, long data2)	None
70	Adds two values and the T bit and refers to the carry	int ovf_addc(long data1, long data2)	None
71	Adds two values and sets the carry to the T bit	long addv(long data1, long data2)	None
72	Adds two values and refers to the carry	int ovf_addv(long data1, long data2)	None
73	Subtracts data2 and the T bit from data1, and sets the borrow to the T bit	long subc(long data1, long data2)	None
74	Subtracts data2 and the T bit from data1, and refers to the borrow	int unf_subc(long data1, long data2)	None
75	Subtracts data2 from data1, and sets the borrow to the T bit	long subv(long data1, long data2)	None
76	Subtracts data2 from data1, and refers to the borrow	int unf_subv(long data1, long data2)	None
77	Subtracts data and the T bit from 0, and sets the borrow to the T bit	long negc(long data)	None
78	Performs division data1/data2 for one step, and sets the result to the T bit	unsigned long div1(unsigned long data1, unsigned long data2)	None
79	Performs initial settings for signed division data1/data2, and refers to the T bit	int div0s(long data1, long data2)	None
80	Performs initial settings for unsigned divisions	void div0u(void)	None
81	Rotates data to left by one bit, and sets the bit pushed out of the operand to the T bit	unsigned long rotl(unsigned long data)	None

82	Rotates data to right by one bit, and sets the bit pushed out of the operand to the T bit	unsigned long rotr(unsigned long data)	None
83	Rotates data including the T bit to left by one bit, and sets the bit pushed out of the operand to the T bit	unsigned long rotcl(unsigned long data)	None
84	Rotates data including the T bit to right by one bit, and sets the bit pushed out of the operand to the T bit	unsigned long rotcr(unsigned long data)	None
85	Shifts data to left by one bit, and sets the bit pushed out of the operand to the T bit	unsigned long shll(unsigned long data)	None
86	Shifts data logically to right by one bit, and sets the bit pushed out of the operand to the T bit	unsigned long shlr(unsigned long data)	None
87	Shifts data arithmetically to right by one bit, and sets the bit pushed out of the operand to the T bit	long shar(long data)	None
88	Performs signed saturation operation for 1-byte data	long clipsb(long data)	None
89	Performs signed saturation operation for 2-byte data	long clipsw(long data)	None
90	Performs unsigned saturation operation for 1-byte data	unsigned long clipub(unsigned long data)	None
91	Performs unsigned saturation operation for 2-byte data	unsigned long clipuw(unsigned long data)	None
92	Sets data to TBR	void set_tbr(void *data)	None
93	Refers to the TBR value	void *get_tbr(void)	None
94	Clears the RB and BL bits of SR to 0, sets the imask value in the I0 to I3 bits of SR, and calls the func function	void sr_jsr(void *func, int imask)	None
95	Sets 1 to the specified bit (bit_num) of the specified address (addr)	void bset(unsigned char *addr, unsigned char bit_num)	void __set1(unsigned char *a, long bit)
96	Sets 0 to the specified bit (bit_num) of the specified address (addr)	void bclr(unsigned char *addr, unsigned char bit_num)	void __clr1(unsigned char *a, long bit)
97	Sets the value of bit [1] (from_bit_num) of address [1] (from_bit_num) to the T bit and bit [2] (to_bit_num) of address [2] (to_addr)	void bcopy(unsigned char *from_addr, unsigned char from_bit_num, unsigned char *to_addr, unsigned char to_bit_num)	None
98	Sets the inverted value of bit [1] (from_bit_num) of address [1] (from_bit_num) to the T bit and bit [2] (to_bit_num) of address [2] (to_addr)	void bnotcopy(unsigned char *from_addr, unsigned char from_bit_num, unsigned char *to_addr, unsigned char to_bit_num)	None

Section 4 Extended Language Specifications

The following SHC #pragma directives are extended specifications specific to the SHC.
If any of the #pragma directives is used in a code, the CC-RH will output a warning and ignores it as an invalid specification.

```
#pragma abs16
#pragma abs20
#pragma abs28
#pragma abs32
#pragma regsave
#pragma noregsave
#pragma noregalloc
#pragma ifunc
#pragma tbr
#pragma gbr_base
#pragma gbr_base1
```

For such a #pragma directive used in the CC-RH, the following warning message will be output at compilation.
Output of this message can be stopped by specifying the -Xcheck=shc option.

W0520161:Unrecognized #pragma.

The following SHC #pragma directives may affect program execution; when the -Xcheck=shc option is specified, a warning message will be output if any of these #pragma directives is used. When a warning message is output, check the use of the #pragma directive shown in the message.

```
#pragma section
#pragma stacksize
#pragma entry
#pragma global_register
#pragma address
```

The following message will be output.

W0523042:Using "function item" function at influence the code generation of "SuperH" compiler

No	Description	SHC	CC-RH
1	Switches sections	#pragma section	The format of this directive is extended in the CC-RH, but the directive can be used in the same format as in the SHC. Note: If the section name specified in the SHC format is the same string as an attribute string in the CC-RH, the specified string is handled as an attribute string. Note that when the -Xcheck=shc option is specified, a message will be output if this directive is handled as "#pragma section *attribute-strings*".
2	Creates an interrupt function	#pragma interrupt	Modify the directive appropriately in accordance with the CC-RH specifications.
3	Performs inline expansion of functions	#pragma inline	This directive can be used in the same format as in the SHC.
4	Expands an assembly-language description function	#pragma inline_asm	This directive can be used in the same format as in the SHC. However, the assembly-language descriptions in #pragma inline_asm should be modified.
5	Branch destination addresses in the specified function are placed on 4-byte boundaries	#pragma align4	Modify the directive appropriately in accordance with the CC-RH specifications.
6	Switches the order of bit assignment	#pragma bit_order	This directive can be used in the same format as in the SHC.
7	Specifies the boundary alignment value for structures, unions, and classes	#pragma pack #pragma unpack	The format of this directive is extended in the CC-RH, but the directive can be used in the same format as in the SHC.

Section 5 Program Compatibility

To use C/C++ source files coded for the SHC as source files for the CC-RH, use the -Xcheck=shc option to check the option settings and extended language specifications that will affect the compatibility. For unspecified behavior and implementation-defined items, use appropriate options such as -Xmisra2004 and -Xmisra2012 to ensure that the source files generate no errors.

5.1 Endian

The CC-RH handles data in little endian, and the data allocation differs from that in the SHC, which is in big endian. For details of data allocation, refer to "4.1.6 Internal representation and value area of data" in the CC-RH Compiler User's Manual. In addition, note the following cases where the difference in endian will cause problems in operation.

(1) Access using a pointer to a type that differs from the defined type

```
1: int val = 0x12345678;
2: void func( void )
3: {
4:   char *p;
5:   ・・・
6:   p = (char *)&val;
```

* On line 6, int-type variable "val" is casted to a pointer to a char type. The difference in endian will cause a problem in operation. Casting to a pointer to a different type can be checked by using the -Xmisra2004 option in the CC-RH.

(2) Access to a union member having a different type

```
1: union {
2:     int u1;
3:     struct {
4:         char  s1;
5:         char  s2;
6:         short s3;
7:     }st;
8: }tag;
```

* For example, when the value 0x01020304 specified in tag.u1 is referenced through a member of tag.st, the read values will differ as follows:
  Big endian:
  tag.st.s1 -> 0x01
  tag.st.s2 -> 0x02
  tag.st.s3 -> 0x0304
  Little endian:
  tag.st.s1 -> 0x04
  tag.st.s2 -> 0x03
  tag.st.s3 -> 0x0102
  Whether unions are used in source files can be checked by using the -Xmisra2004 option in the CC-RH.

(3) Access to a variable that has an initial value allocated in different endian
Before accessing it, convert the ROM data allocated in big endian to little endian or use the CC-RH intrinsic function __bsw() or __bsh() to convert the endian of the variable.

5.2 Programs out of ANSI Specifications

The following programs can be successfully compiled in the SHC, but they will generate errors in the CC-RH. These settings and definitions are outside the ANSI specifications; check and modify them.

(1) Assignment of a constant whose value is outside the range of the variable

- Example

```
int a = 12345678901234567890123456789012345678901234567890;
```

Modify the value of the constant assigned to a variable so that it fits into the range representable by the data type of the variable.

- Modification example

```
int a = 12345678901234567890;
```

(2) Escape sequence as a character constant

- Example

```
char *x = “¥xh”;
```

¥x<hexadecimal number> indicates a hexadecimal integer. For a constant that is not a hexadecimal number, delete ¥.

- Modification example

```
char *x = “xh”;
```

(3) Definition of an enumerator having a value that cannot be represented by the int type

- Example

```
1: #include <limits.h>
2: enum { A = INT_MAX, B }
```

Modify the value of an enumeration constant so that it fits into the range representable by an int-type constant.

- Modification example

```
1: #include <limits.h>
2: enum { A = INT_MAX, B = 0x80000000U };
```

(4) Illegal characters in a preprocessor directive

- Example

```
1: #if x ! y == 3
2: #endif
```

Delete the illegal characters in a preprocessor directive.

- Modification example

```
1: #if x
2: #endif
```

(5) Assignment to a conditional operator when the conditional expression is a constant

- Example

```
1: void main() {
2:     char *p;
3:     * (1 ? (char *)p : (const char *)p) = 0;
4: }
```

A conditional operator is not allowed on the left side of an assignment expression. Assign a value indirectly as shown in the following modification example.

- Modification example

```
1: void main() {
2:     char *p;
3:     char *p2;
4:     p2 = (1 ? (char *)p : (const char *)p);
5:     *p2 = 0;
6: }
```

(6) Unmatched function declaration

- Example

```
1: void main() {
2:     { int f(int, ...); int f(); }
3: }
```

When multiple functions are declared with the same name but have different numbers of parameters or different types of parameters, delete the incorrect ones.

- Modification example

```
1: void main() {
2:     { int f(int, ...); }
3: }
```

(7) Illegal identifier in a unary operator "defined" expression

- Example

```
1: #if defined (x
2: #endif
```

For the above preprocessor directive, use the form of "#if defined (*identifier*)" or "#if defined *identifier*"

- Modification example

```
1: #if defined (x)
2: #endif
```

(8) sizeof for an incomplete variable

- Example

```
1: void main(){
2:     extern int i[]; {
3:         extern int i[10];
4:     }
5:     sizeof(i);
6: }
```

An incomplete variable must not be passed as the argument for sizeof operation. Modify as follows.

- Modification example

```
1: void main()
2: {
3:     extern int i[]; {
4:         extern int i[10];
5:         sizeof(i);
6:     }
7: }
```

Section 6 Assembler Directives and Macro Function

6.1 Assembler Directives

The following table shows the correspondence between the assembler directives in the SHC and the CC-RH.

Type	Description	SHC	CC-RH
Target CPU	Specifies the target CPU	.CPU	None
Section and location counter	Declares a section	.SECTION	.cseg Indicates the start of a code section. .dseg Indicates the start of a data section. .org Indicates the start of a section at an absolute address. The boundary alignment value of the start address of the section cannot be specified.
	Sets the value of the location counter	.ORG	None
	Corrects the value of the location counter to a multiple of the boundary alignment value	.ALIGN	.align
Symbols	Sets a symbol value	.EQU	.equ The externally referenced symbols cannot be specified.
	Sets or resets a symbol value	.ASSIGN	.set
	Defines the alias of a register name	.REG	None
	Defines a floating-point register name	.FREG	None
Data and data area reservation	Reserves integer data	.DATA	.db Initializes a byte area. .db2/.dhw Initializes a 2-byte area. .db4/.dbw Initializes a 4-byte area.
	Reserves an integer data block	.DATAB	None
	Reserves string literal data	.SDATA	None
	Reserves a string literal data block	.SDATAB	None
	Reserves string literal data (with length)	.SDATAC	None
	Reserves string literal data (with zero terminator)	.SDATAZ	.db
	Reserves a floating-point data area	.FDATA	.float Initializes a 4-byte area. .double Initializes an 8-byte area.
	Reserves a floating-point data block	.FDATAB	None
	Reserves fixed-point data	.XDATA	None
	Reserves a data area	.RES Reserves a memory area with the size specified	.ds If a size is specified, the compiler will fill the specified number of bytes
	Reserves a string literal data area	.SRES	None
	Reserves a string literal data area (with length)	.SRESC	None
Data and data area reservation	Reserves a string literal data area (with zero terminator)	.SRESZ	None
	Reserves a floating-point data area	.FRES	None

Externally defined and externally referenced symbols	Declares externally defined symbols	.EXPORT	.public
	Declares externally referenced symbols	.IMPORT	.extern
	Declares externally defined and externally referenced symbols	.GLOBAL	None
Object modules	Controls output of the object module and debugging information	.OUTPUT	None
	Controls the output of symbolic debugging information	.DEBUG	None
	Selects big endian or little endian	.ENDIAN	None
	Changes the line number	.LINE	None
Assemble listing	Controls output of the assemble listing	.PRINT	None
	Controls output of the source program listing	.LIST	None
	Sets the number of lines and columns in the assemble listing	.FORM	None
	Sets the header for the source program listing	.HEADING	None
	Inserts a new page in the source program listing	.PAGE	None
	Outputs blank lines to the source program listing	.SPACE	None
Other directives	Sets the name of the object module	.PROGRAM	None
	Sets the radix (base) for integer constants with no radix specification	.RADIX	None
	Specifies an entry point and the end of the source program	.END	None
	Defines the stack value for the specified symbol	.STACK	.stack

6.2 Macro Function

The following table shows the correspondence between the macro function directives in the SHC and the assembler directives in the CC-RH.

Description	SHC	CC-RH
Defines a macro instruction	.MACRO - .ENDM	.macro - .endm
Terminates macro instruction expansion	.EXITM	.exitm Skips out of the current repetitive assembly loop. .exitma Skips outside all the nested repetitive assembly loops.

Section 7 Others

The SHC and CC-RH specifications differ in the following functions.

(1) Section address operator

__sectop, __secend, and __secsize can be used in the SHC. In the CC-RH, use the following section set operators in assembly-language codes instead of them.

- STARTOF
- SIZEOF

(2) enum type

When an undefined identifier is specified in an enum type, the undefined identifier is set to 0 in the SHC, but the CC-RH will generate an error.

End of document