To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Date: Jun.07.2004

RENESAS TECHNICAL UPD

Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan
RenesasTechnology Corp.

Product Category	User Development Environment		Document No.	TN-CSX-073A/EA	Rev.	1.0
Title	SuperH RISC engine C/C++ compiler package V.8.00 Release 02 Updates		Information Category	Specification Change		
Applicable Product	P0700CAS8-MWR, R0C40700XSW08R P0700CAS8-SLR, R0C40700XSS08R P0700CAS8-H7R, R0C40700XSH08R	Lot No. All	Reference Document	SuperH RISC engine C/C++ Compiler Assembler Optimizing Linkage Editor User's Manual REJ10B0047-0100H Rev.1.00		

SuperH RISC engine C/C++ compiler package is updated in V.8.00 Release 02.

See the notes below if you have the compiler package listed in the following table.

Part No.	Package version	Compiler version
P0700CAS8-MWR	8.0.00 8.0.01	8.0.00 8.0.01
R0C40700XSW08R	8.00 Release 01	8.0.01
P0700CAS8-SLR	8.0.00 8.0.01	8.0.00 8.0.01
R0C40700XSS08R	8.00 Release 01	8.0.01
P0700CAS8-H7R	8.0.00 8.0.01	8.0.00 8.0.01
R0C40700XSH08R	8.00 Release 01	8.0.01

If you have the compiler package of the Windows® version, download the update program from the following URL:

http://www.renesas.com/eng/products/mpumcu/tool/index.html

If you have the compiler package of the UNIX version, request the update program to an authorized product distributor.

The contents of updates in this package are shown below.

Descriptions of sections 1 and 2 only apply to the Windows® version.

1. High-performance Embedded Workshop (Windows® version)

1.1 Improvement of Workspace window Look&Feel

The [Projects] tab in [Workspace] window can show the files as time stamp order. Out of date files (those updated after the previous build) can be marked in the [Workspace] window.

1.2 Enhancement for Makefile Generation

HEW is now capable of generating makefile for GNUMake. As well as makefile is for Hmake and Nmakes. This allows use of a general-purpose make tool that supports GNUMake. Compiler options can be output to separate files (sub-command files).

1.3 Customization of the HEW Linkage Order

Customization of the HEW linkage order is newly supported.

1.4 Virtual Desktop Function

The virtual desktop function is newly supported. It is possible to have a maximum of four window configurations so that users are able to use the screen effectively by switching these window configurations.

1.5 Enhancement of saving view contents

The contents of the view Cache (SH only), I/O, PA, Register, and TLB (SH only) can be saved into text file.

1.6 Specifications of the [Watch] Window Changed

Variables added in the [Watch] window will be retained even after the window is closed, unless the user deletes these variables manually.

1.7 Improved Tools Options Dialog Box

The size of the [Section] dialog box opened from the Tools Options Dialog Box is now customizable.

1.8 Direct Display of the Source File at the Current PC

This function (toolbar button) allows the source file at the current PC to be displayed.

1.9 Enhanced Downloading Function

The following new options can be specified:

(1) Before downloading a load module, the HEW checks if the source file has been modified. If modified, the HEW automatically builds the module before downloading.

(2) After downloading the load module, the HEW automatically resets the target program.

1.10 Improved Address Field

An address field now has a function to refer to the list of labels. The last 20 items entered in the address field can be shown in the drop-down list.

1.11 Auto-Recovery Function

This function is newly supported to backup the workspace, project, and session files at regular intervals.

1.12 Enhanced Function to Customize the Display Format

The function to customize fonts and size has been enhanced.

1.13 Freeze while Editing

We have corrected the problem of the HEW being halted while editing a file in the editor with the navigation facility enabled.

1.14 Illegal Termination of the HEW after Adding a Custom Build Phase

We have corrected the problem of the HEW illegally terminated by pressing the [OK] button after specifying an output file in [(Phase name) Options] for the added custom phase.

1.15 Duplicate Header File Name in the [Workspace] Window

We have corrected the problem of displaying a duplicate header file name on the [Project] tab of the [Workspace] window when the header file was defined with both uppercase and lowercase letters.

[Example]

File1.c: #include "SAMPLE.H"

File2.c: #include "sample.h"

1.16 Incorrect Display of Navigation

We have corrected the problem of incorrectly displaying information on the [Navigation] tab of the [Workspace] window when a space (') was attached to the number of elements in an array-type variable declaration.

[Example]

extern int tbl [2]

1.17 Incorrect Dependencies for a Custom Build Phase

We have corrected the problem of changing the names of files with dependent information on the custom build phase by adding a file to the project after the custom build phase had been added to the said project.

2. SuperH RISC engine simulator/debugger (Windows® version)

2.1 Setting the memory read cycles and write cycles

Both number of memory read cycles and write cycles can be modified.

3. Compiler

3.1 Illegal Copy Propagation

We have corrected the problem of illegally eliminating a copy instruction when the copy instruction existed in a block with multiple branch sources.

[Example]

```
int func(int *x) {
    int ret=0;
    while(*x++){
        if(*x==1){
            ret+=2;
        }
    }
    return (ret+2);
}

_func:
    MOV     #0,R5    ; Illegally eliminates the copy instruction and converts R7 to R5
L11:
    MOV.L   @R4,R2
    ADD     #4,R4
                ; *1 Illegally eliminates MOV R7,R5
```

```
        TST     R2,R2
        ADD     #2,R5
        BT      L13
        MOV.L   @R4,R0
        CMP/EQ  #1,R0
        BT      L11     ; *2 By *3, BF L11 is converted
        BRA     L11
        NOP             ; *3 Illegally eliminates MOV R5,R7
    L13:
        RTS
        MOV     R5,R0
```

[Conditions]

This problem might occur when all of the following conditions were fulfilled.

(1) optimize=1 was specified.

(2) A conditional statement was described.

(3) A copy instruction existed in a block with multiple branch sources (*1 in the above example).

(4) The block of the branch sources in (3) had a path with no definition of the copy source register (R7 in the above example) for the copy instruction (in the example, the path branching from *2 to L11).

3.2 Illegal Elimination of Unnecessary Expressions

We have corrected the problem of illegally eliminating a conditional statement if a then or else clause of the conditional statement had an assignment expression and another assignment expression, of which the both sides had the same variable, followed the said expression.

[Example]

```
int x;
void f(int y){
    if (y>=256){    /* Illegal elimination */
        x=0;        /* *1 */
    }
    x=x;            /* *2 Eliminates the assignment expression that has the same */
                    /*    variable in both sides                                 */
    x++;
}
```

↓

```
void f(int y){
    x=0;
    x++;            /* Propagates x=0 */
}
```

↓

```
void f(int y){
    x=1;
}
```

[Conditions]

This problem might occur when all of the following conditions were fulfilled.

(1) optimize=1 was specified.

(2) A conditional statement was described.

(3) A then or else clause of the conditional statement of (2) had an assignment expression (*1 in the above example).

(4) An assignment expression, in which the both sides had the same variable as the variable assigned to in (3), followed the conditional statement of (2) (*2 in the above example).

3.3 Illegal Access with a Parameter Passed via the Stack

We have corrected the problem of an address for reference to a parameter passed via the stack being incorrect when the speed option was specified if a function with the parameter passed via the stack had a function call immediately before the exit.

[Example]

```
typedef struct {
    int x;
} ST;
extern void g(ST *x);
void f(int a, ST b) {  /* b is a parameter passed via the stack */
    if (a) {
        g(&b);
        /* (A) */
    }
    /* (B) */
}


                    ; Address where parameter b is stored at the function entry = R15
_f:
    TST     R4,R4
    BT      L12
    MOV     R15,R4
    MOV.L   L14,R2  ; _g
    JMP     @R2     ; (A)
    ADD     #4,R4   ; R4 <- R15+4 : Not the address of b
L12:
    RTS             ; (B)
    NOP
```

[Conditions]

This problem might occur when all of the following conditions were fulfilled.

(1) optimize=1 was specified.

(2) The speed option was specified.

(3) The function had a parameter passed via the stack (b in the above example).

(4) The function had multiple exits ((A) and (B) in the above example).

(5) There was a function call immediately before any of the exits in (4) (g(&b); in the above example).

(6) (5) was the only function call in this function.

3.4 Illegal GBR Relative Logic Operation

We have corrected the problem of writing the result to an incorrect area if a logic operation with a 1-byte array or a bit-field member, for which #pragma gbr_base/gbr_base1 was specified, was performed.

[Example]

```
#pragma gbr_base a,b
char a[2],b[2];
void f() {
    a[0] = b[0] & 1;
}


    MOV     #_b-(STARTOF $G0),R0
    RTS
    AND.B   #1,@(R0,GBR)            ; Writes the result of the operation to b[0]
```

[Conditions]

This problem might occur when all of the following conditions were fulfilled.

(1) gbr=user was specified.

(2) #pragma gbr_base/gbr_base1 was specified for any of the following variables:

- An (unsigned)char-type array
- A structure array that has an (unsigned)char-type member
- A structure that has an (unsigned)char-type array member
- A structure that has a bit-field member of 8 bits or less

(3) A logic operation of a constant (&, |, ^) with the variable of (2) (b[0] in the above example) was performed.

(4) The variable assigned to by the operation of (3) (a[0] in the above example) fulfilled the condition of (2).

(5) Variables of (3) and (4) were different variables, different elements of the same array, or different members of the same structure.

3.5 Illegal Elimination of Sign/Zero Extension

We have corrected the problem of eliminating the cast when the address of a variable/constant or the index of an array was cast to 1 or 2 bytes and this value was used for accessing memory, or the expression which was cast to a char type was assigned to an unsigned short type variable and the result was used for comparison.

[Example1]

```
unsigned short x;
char a[1000];


void f() {
    a[(char)x] = 0;
}


    MOV.L   L11+2,R2        ; _x
    MOV.L   L11+6,R6        ; _a
```

```
        MOV.W    @R2,R5
        EXTU.B   R5,R0
                                ; Eliminates EXTS.B R0,R0
        MOV      #0,R5              ; H'00000000
        RTS
        MOV.B    R5,@(R0,R6)       ; When x is not within the range of 0 to 127,
                                ; an illegal address may be referred to.
```

[Example2]

```
  unsigned short us0;
  unsigned int b;

  func1() {
      unsigned short us1;
      us1 = (char)b;
      return(us0 !=us1);
  }

      MOV.L    L11,R2             ; _b
      MOV.L    L11+4,R5           ; _us0
      MOV.L    @R2,R6
      EXTS.B   R6,R2
      MOV.W    @R5,R6
      EXTU.W   R6,R5
      CMP/EQ   R2,R5              ; (char)b is not cast to an unsigned short type
                                ; and is used in comparison.
      MOVT     R0
      RTS
      XOR      #1,R0
```

[Conditions]

This problem might occur when all of the following conditions were fulfilled.

(1) optimize=1 was specified.

(2) One of the following conditions (a)(b) was satisfied.

(a-1) The address of a variable/constant or the index of an array was explicitly cast to 1 or 2 bytes.

(a-2) The value of (a-1) was used for accessing memory.

(b-1) The expression which was cast to a char type was assigned to an unsigned short type variable.

(b-2) The variable of (b-1) was used in comparison.

3.6 Former Limitation

We have corrected the following problem:

When addition and subtraction of a section address operator and a numeric value were described to the initial value of an external variable, the portion of numerical addition and subtraction was not output in an object.

[Example]

```
char *addr1 = (char *)__sectop("P");                    // OK
char *addr2 = (char *)__sectop("P") + __secsize("P"); // OK
char *addr3 = (char *)__sectop("P") + 10;               // NG
```

3.7 Internal Errors Corrected

We have corrected the problem of generating internal errors under the following conditions:

(1) A C++ program, which only had declaration of a class that includes a static function member declaration using the function-type typedef, was compiled with the debug option specified.

(2) A C/C++ program, which includes a switch statement that is not enclosed by {} and had no default label, was compiled with optimize=0 and the debug option specified.

4. Optimizing Linkage Editor

4.1 Illegal output data for an empty area in S-Type/HEX format files

The following problem had been fixed:

When an output file format is S-Type (or HEX), "0" character (ASCII-code:0x30) might be incorrectly changed to NULL (ASCII-code:0x00) at an empty area in which no instruction or data exists.

[Condition]

This problem might occur when all of the following conditions were fulfilled.

(1) The endian=little option was specified for the compiler (or assembler).

(2) The form=stype(or hexadecimal) option was specified for the linker.

(3) The code (or data) linked to the same section are divided and exist in two or more input object files.

(4) "0" was inserted in the section of (3) by boundary adjustment at the linkage process.

4.2 Illegal object due to specification of unifying same codes optimization

Fixed the problem that a branch to the unified subroutine might be incorrect.

[Condition]

This problem might occur when all of the following conditions were fulfilled.

(1) Unifying same codes optimization (optimize=same_code) was valid.

(2) The input file was compiled with the goptimize option.

(3) Two or more unified subroutines were generated by optimization.

(4) A distance between the generated subroutines of (3) was 4096 bytes or less.

4.3 Incorrect error with specification of the change_message option

Fixed the problem that an incorrect error occurred when two or more error levels were specified in the change_message option.

[Example]

When the change_message option was specified as follows, a linker older than version 8.0.03 output an error message incorrectly:

```
optlnk -change_message=e=1000,w=2000 *.obj
```