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.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RX Family C/C++ Compiler Package

Application Notes: RX Migration Guide, H8 Edition

This document explains the items that need to be checked for migration from H8 family source program, for C/C++ Compiler V1 for the RX family.

Table of contents

Introduction 2

1. Options 3
1.1 Specifying sign for the char type 3
1.2 Specifying sign for bit field members 5
1.3 Specifying bit field member allocation 6
1.4 Specifying endian 7
1.5 Specifying the size of double type variables 8
1.6 Specifying the size of int type variables 9

2. Language specification 10
2.1 Signs for char types 10
2.2 Specifying sign for bit field members 11
2.3 Endian 12
2.4 Size of double type variables 13
2.5 Size of int type variables 14
2.6 asm blocks 15

3, Optimization option setting for migration from H8-family 16

Exsample: Sample source 17

Web site and support <website and support> 18

Introduction

In this application notes, it explains the software migration method when C program made by H8SX, H8S, H8 family compiler (It is recorded as H8-family) is transplanted to RX-family compiler.

In Renesas RX-family compiler, the function to absorb the diference between the option and the language specification is supported inconsideration of the migration from H8-family to RX-family. As a result, the application part of the embedded sofutware can be smoothly transplanted.

In this application notes, it explains the use of the function that the difference between the option and the language specification of the compiler for H8-family and the RX-family and compilers for the RX-family support it.

Please use this application notes when you transplanted to RX-family from H8-family.

1. Options

Some specifications differ for the default options between H8-family compilers and RX-family compilers. The following explains options that will likely require handling during migration from H8 to RX.

Note that the expansion code in assembly code as used in this document can be obtained by specifying “output=src” and “cpu=rx600”.

When the “cpu” option is different, the expansion code in assembly language may also differ. Also, the expansion code in assembly language may change due to subsequent compiler improvements, so please use this for reference.

No	Functionality	H8 option	RX option	Reference
1	Specifying sign for the char type	–	signed_char	1.1
2	Specifying sign for bit field members	–	signed_bitfield	1.2
3	Specifying bit field member allocation	bit_order	bitorder=left	1.3
4	Specifying endian	–	endian=big	1.4
5	Specifying the size of double type variables	double=float	dbl_size=8	1.5
6	Correspondence of int type variable size to difference	–	int_to_short	1.6

1.1 Specifying sign for the char type

H8-family compilers treat char types with no sign specified as signed char types, whereas RX-family compilers treat them as unsigned char types in default. To migrate to RX a program created in H8 based on the requirement that char types be signed char types, specify the “signed_char” option.

Format

signed_char : unsigned_char by default
unsigned_char

[How to set this option in the Renesas IDE]

Choose Build and then RX Standard Toolchain, and perform the following settings in the displayed dialog box.

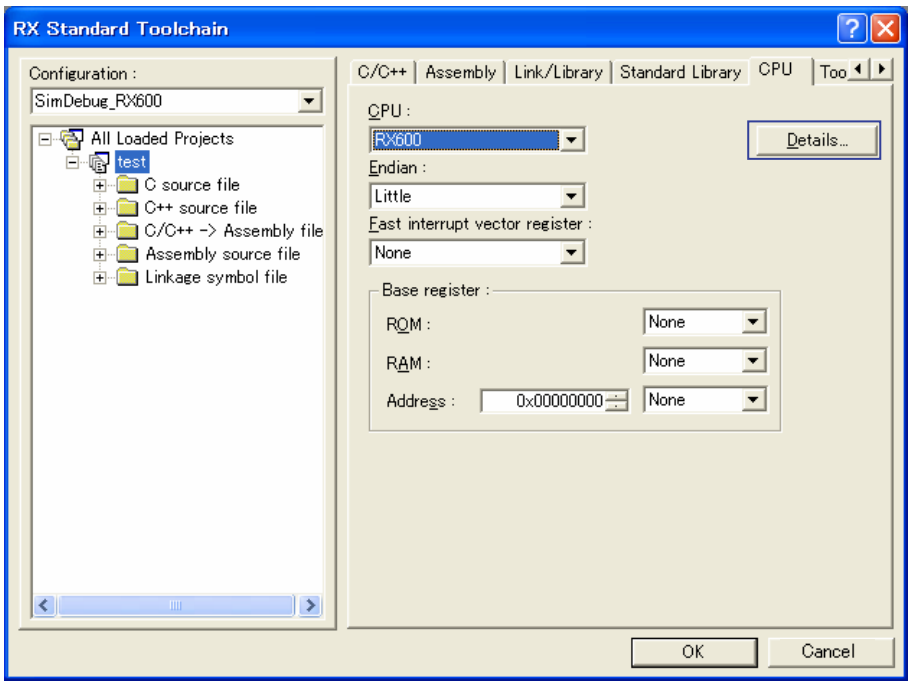

Figure 1-1

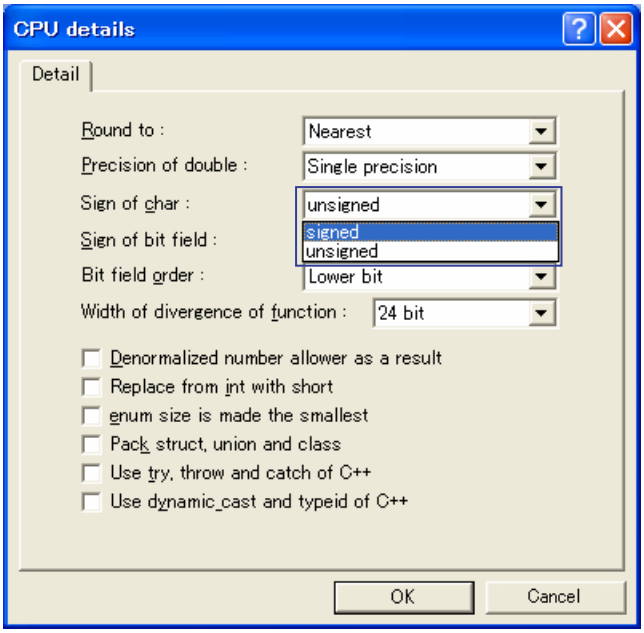

Figure 1-2

1.2 Specifying sign for bit field members

H8-family compilers treat bit field members with no sign specified as signed types, whereas RX-family compilers treat them as unsigned types in default.

To migrate to RX a program created in H8 based on the requirement that bit field members with no sign specification are signed types, specify the “signed_bitfield” option.

Format

signed_bitfield : unsigned_bitfield by default
unsigned_bitfield

[How to set this option in the Renesas IDE]

In RX Standard Toolchain (Figure 1-1), choose Details, and perform the following settings in the displayed dialog box.

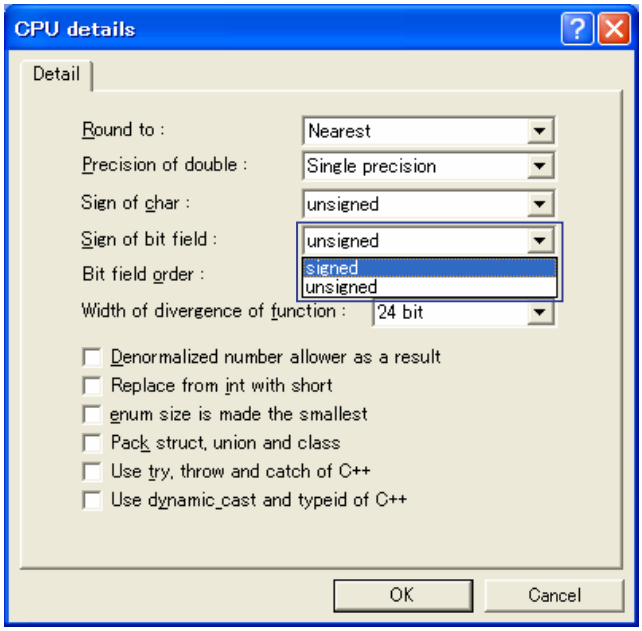

Figure 1-3

1.3 Specifying bit field member allocation

With H8-family compilers, bit field members are allocated from the most significant bit, whereas with RX-family compilers, they are allocated from the least significant bit in default. To migrate to RX a program created in H8 based on the requirement that bit field members are allocated from the most significant bit, specify the “bit_order=left” option.

Format

bit_order={left|right} : right by default

[How to set this option in the Renesas IDE]

In RX Standard Toolchain (Figure 1-1), choose Details, and perform the following settings in the displayed dialog box.

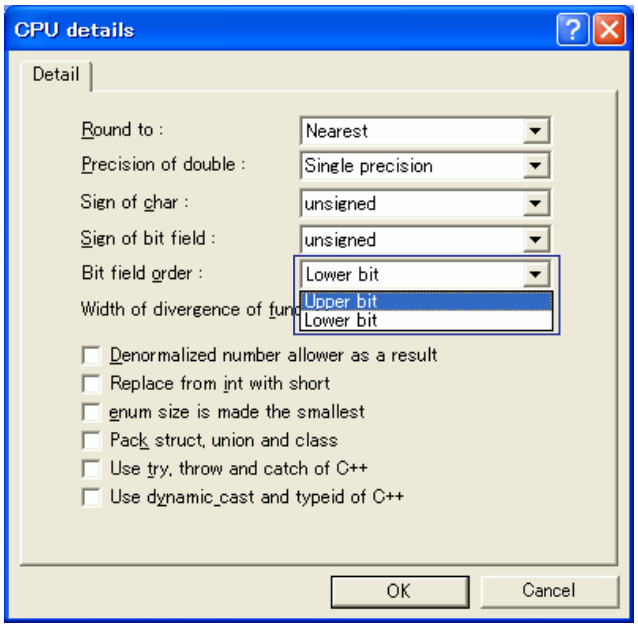

Figure 1-4

1.4 Specifying endian

With H8-family compilers, the data byte order is big-endian, whereas with RX-family compilers, it is little-endian in default.

To migrate to RX a program created in H8 based on the requirement that the data byte order is big-endian, specify the "endian=big" option.

Format

endian={big|little} **: little by default**

[How to set this option in the Renesas IDE]

Choose Build and then RX Standard Toolchain, and perform the following settings in the displayed dialog box.

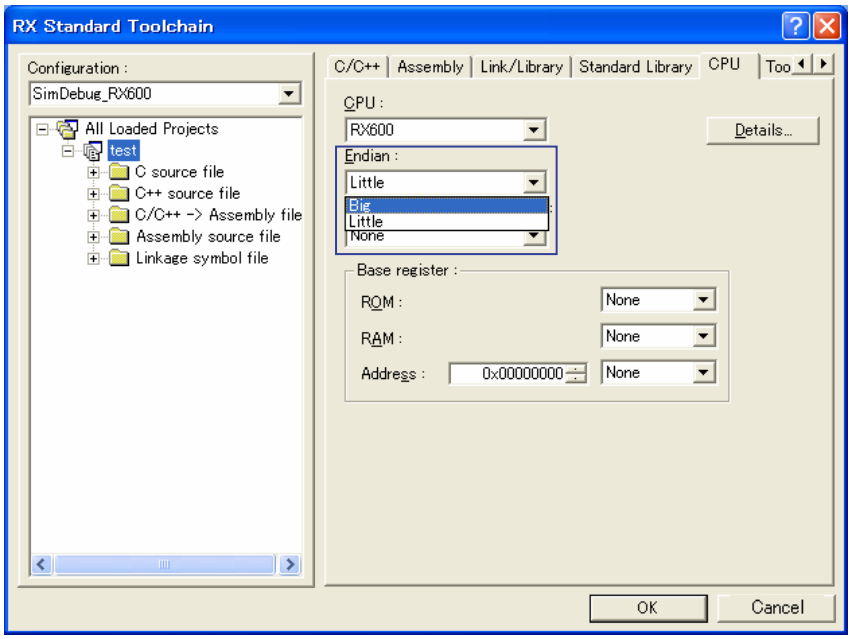

Figure 1-5

1.5 Specifying the size of double type variables

With H8-family compilers, the size of the double type is 8 bytes, whereas with RX-family compilers, the size of the double type is four bytes in default . To migrate to RX a program created in H8 based on the requirement that the size of the double type is 8 bytes, specify the “dbl_size=8” option.

Format

dbl_size ={4|8} **: 4 by default**

[How to set this option in the Renesas IDE]

In RX Standard Toolchain (Figure 1-1), choose Details, and perform the following settings in the displayed dialog box.

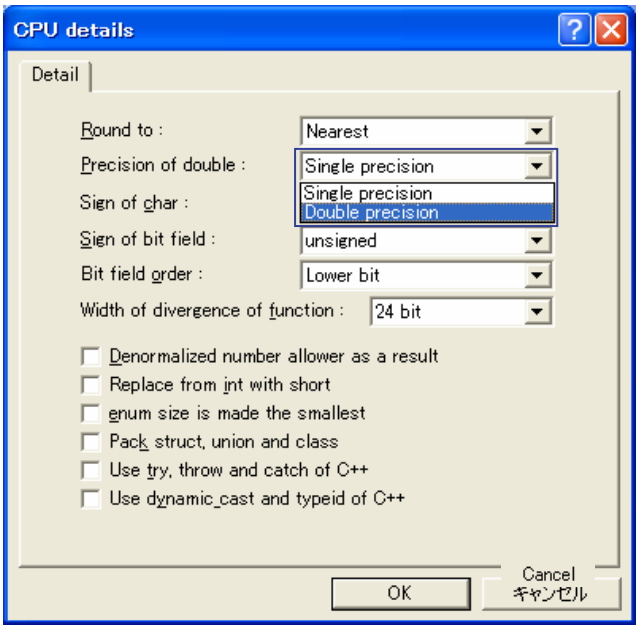

Figure 1-6

Precaution:

In optional double=float of the compiler for the H8 family, the size of the float type and the double type is four bytes.

And the long long type is eight bytes.

In optional dbl_size=4 of the compiler for the RX family, it is four bytes as for the size of the float type, the double type, and the long double type.

The result of conversion/library related to the floating point might be different from the result of the compiler for the H8 family when optional dbl_size=4 is effective.

1.6 Correspondence of int type variable size to difference

With H8-family compilers, the size of the int type is 2 bytes, whereas with RX-family compilers, the size of the int type is 4 bytes in default. To migrate to RX a program created in H8 based on the requirement that the size of the int type is 2 bytes, specify the “int_to_short” option.

Format

int_to_short

[How to set this option in the Renesas IDE]

In RX Standard Toolchain (Figure 1-1), choose Details, and perform the following settings in the displayed dialog box.

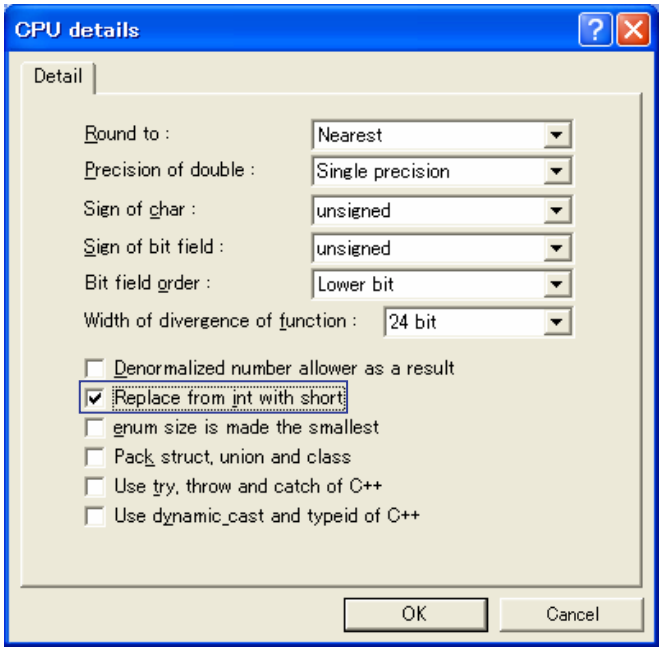

Figure 1-7

Precaution:

- The value of the following macros in imits.h doesn't change even if it uses optional int_to_short.
  INT_MAX, INT_MIN, and UINT_MAX.
- The variable and the function that is declared in the int type and defined are replaced with the short type and it compiles.
  When a pan-integral promotion of the comparison type etc. is done, the variable enhances to the type in four bytes and evaluates the value.
- This option cannot be specified by combining with lang=cpp or lang=ecpp. Optional int_to_short becomes invalid when combining.

For exsample:

```
char a= -1;
if ( a < 0xFF )
```

The truth and the imitation are different by RX and H8.

2. Language specification

This chapter explains the language specifications that need to be changed during RX migration.

Table 2-1 List of language specifications

No	Functionality	Reference
1	Signs for char types	2.1
2	Specifying sign for bit field members	2.2
3	Endian	2.3
4	Size of double type variables	2.4
5	Size of int type variables	2.5
6	asm blocks	2.6

2.1 Signs for char types

With H8-family compilers, char types without a specified sign are treated as signed char types, whereas with RX-family compilers, they are treated as unsigned char types in default. When programs created in H8 based on the requirement that char types are signed char types are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to absence/presence of sign for the char type

Source code

```
char a = -1;

void main(void)
{
    if (a < 0) {
        // char types are signed and 'a' is interpreted as negative,
        // so the expression is satisfied (H8)
    } else {
        // char types are unsigned and 'a' is interpreted as positive,
        // so the expression is not satisfied (RX)
    }
}
```

To migrate to RX a program created in H8 based on the requirement that a signed char type is used for the char type, specify the "signed_char" option. For details about how to specify this option, see *1.1 Specifying sign for the char type.*

2.2 Specifying sign for bit field members

With H8-family compilers, bit field members with no sign specified as treated as signed types, whereas with RX-family compilers, they are treated as unsigned types in default.

When programs created in H8 based on the requirement that bit field members with no sign specified are signed types are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to absence/presence of sign for bit field members

Source code

```
struct S {
   int a : 15;
} s = { -1 };

void main(void)
{
   if (s.a < 0) {
      // bit field members are signed and 's.a' is interpreted as negative,
      // so the expression is satisfied (H8)
   } else {
      // bit field members are unsigned and 's.a' is interpreted as positive,
      // so the expression is not satisfied (RX)
   }
}
```

To migrate to RX a program created in H8 based on the requirement that a bit field member with no sign specified is a signed type, specify the "signed_bitfield" option. For details about how to specify this option, see *1.2 Specifying sign for bit field members*.

2.3 Endian

With H8-family compilers, the data byte order is big-endian, whereas with RX-family compilers, it is little-endian in default.

When programs created in H8 based on the requirement that the data byte order is big-endian are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to variance for endian

Source code

```
typedef union{
    short data1;
    struct {
        unsigned char upper;
        unsigned char lower;
    } data2;
} UN;

UN u = { 0x7f6f };

void main(void)
{
    if (u.data2.upper == 0x7f && u.data2.lower == 0x6f) {
        // When the data byte order is big-endian (H8)
    } else {
        // When the data byte order is little-endian (RX)
    }
}
```

To migrate to RX a program created based on the requirement that data byte order is big-endian, specify the “endian=big” option. For details about how to specify this option, see *1.4 Specifying endian*.

2.4 Size of double type variables

With H8-family compilers, the size of the double type is 8 bytes, whereas with RX-family compilers, the size of the double type is 4 bytes in default. When H8 programs created based on the requirement that the size of the double type is 8 bytes are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to variance in double type size

Source code

```
double d1 = 1E30;
double d2 = 1E20;

void main(void)
{
    d1 = d1 * d1;       // When the size of the double type is 4 bytes, d1 * d1 overflows
    d2 = d2 * d2;       // When the size of the double type is 4 bytes, d2 * d2 overflows

    if (d1 > d2) {
        // When the size of the double type is 8 bytes,
        // normal size comparison is performed (H8)
    } else {
        // When the size of the double type is 4 bytes,
        // size comparison fails because both d1 and d2 overflow (RX)
    }
}
```

When migrating programs created based on the requirement that the size of the double type is 8 bytes to RX, specify the “dbl_size=8” option. For details about how to specify this option, see *1.5 Specifying the size of double type variables.*

2.5 Size of int type variables

On H8-family compilers, the size of the int type is 2 bytes, whereas on RX-family compilers the size of the int type is 4 bytes in default. When H8 programs created based on the requirement that the size of the int type is 2 bytes are migrated to RX, they may not operate properly.

Example: Code for which operation is different due to variance in int type size

Source code

```
typedef union{
    int data1;
    struct {
        unsigned char upper;
        unsigned char lower;
    } data2;
} UN;

void main(void)
{
    UN u;
    u.data1 = 0x7f6f;

    if (u.data2.upper == 0x7f && u.data2.lower == 0x6f) {
        // When the size of the int type is 2 bytes (H8)
    } else {
        // When the size of the int type is 4 bytes (RX)
    }
}
```

To migrate to RX a program created based on the requirement that the size of the int type is 2 bytes, specify the “int_to_short” option. For details about how to specify this option, see *1.6 Correspondence of int type variable size to* difference.

2.6 asm blocks

H8-family compilers allow asm blocks to be used to code assembly language programs in C source programs. Since RX-family compilers lack the corresponding functionality, programs using asm blocks need special handling when migrated to RX.

RX-family compilers have assembly code functions to code assembly language in C source programs. The contents coded in the asm block can sometimes be handled by being coded in the assembly code function.

For details about assembly code functions, see *1.2 Performing inline expansion in assembly code functions* in C/C++ Compiler Package for the RX Family Application Notes: Compiler Usage Guide, Extended Functionality Edition.

Example:

Program using the H8 asm function and program using the assembly code function in RX

Source code using an H8 asm block

C source code

```
void func(void)
{
   asm {
       NOP
   }
}
```

Assembler source expansion code

```
_func:
   NOP
   rts
```

Source code using the RX assembly code function

C source code

```
#pragma inline_asm asm_nop
static void asm_nop(void)
{
   NOP
}

void func(void)
{
   asm_nop();
}
```

Assembler source expansion code

```
_func:
   NOP
   RTS
```

Precautions

- H8 allows variables to be coded in the assembler, but RX does not.

3, Optimization option setting for migration from H8-family

There is a difference in an optional setting method for optimization in the compiler of H8-family, and RX-family.
Please refer to the following optimization option setting when embedded software transplant from H8-family to RX-family and the performance is evaluated.

Optimization option setting of each compiler and comparison of ROM size

(The sample program for the measurement is described to the next page.)

H8SX	Optimize OFF	Optimize Size			Optimize Speed		
	opt=0	opt=1	-	-	opt=1 speed	-	-
main()	0xCE	0xAA	-	-	0x214	-	-
sort()	0xC0	0x86	-	-	0xA0	-	-

RX	Optimize OFF	Optimize Size			Optimize Speed		
	optimize=0	optimize=1	optimize=2	optimize=max	optimize=1 speed	optimize=2 speed	optimize=max speed
main()	0xD1	0x95	0x6A	0x6A	0x95	0xEB	0x11A
sort()	0xFD	0x69	0x65	0x65	0x6A	0x5F	0x5F

1、 Optimization option setting for migration from H8-family

(1) Please specify it for RX above opt=1 or opt=1 speed when you specify opt=0 for H8-family.
In optimize=0 for RX-family, the meaning is different from opt=0 for H8-family.

Exsample:
・If opt=0 for H8SX is used, rom size is 0xCE (main()) -> If optimize=0 for RX is used, rom size is 0xD1 (main()) …NG
・If opt=0 for H8SX is used, rom size is 0xCE (main()) -> If optimize=1 for RX is used, rom size is 0x95 (main()) … OK

(2) Please specify optimize=2 or optimize=2 speed or more for RX-family when you specify opt=1 or opt=1 speed for H8S-famly.
Opt=1 or opt=1 speed for RX-family is the same meaning as opt=0 for H8-family.
Exsample:
・If opt=1 for H8SX is used, rom size is 0xAA (main()) -> If optimize=1 for RX is used, rom size is 0x95 (main()) … NG
・If opt=1 for H8SX is used, rom size is 0xAA (main()) -> If optimize=2 for RX is used, rom size is 0x6A (main()) …OK

Please refer to the compiler user's manual for details of the optimization level of the RX compiler.

Exsample: Sample source

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void main(void);
void sort(long *a);
void change(long *a);

void main(void)
{
    long a[10];
    long j;
    int i;

    printf("### Data Input ###¥n");

    for( i=0; i<10; i++ ){
        j = rand();
        if(j < 0){
            j = -j;
        }
        a[i] = j;
        printf("a[%d]=%ld¥n",i,a[i]);
    }
    sort(a);
    printf("*** Sorting results ***¥n");
    for( i=0; i<10; i++ ){
        printf("a[%d]=%ld¥n",i,a[i]);
    }
change(a);
}
```

```
void sort(long *a)
{
    long t;
    int i, j, k, gap;

    gap = 5;
    while( gap > 0 ){
        for( k=0; k<gap; k++){
            for( i=k+gap; i<10; i=i+gap ){
                for(j=i-gap; j>=k; j=j-gap){
                    if(a[j]>a[j+gap]){
                        t = a[j];
                        a[j] = a[j+gap];
                        a[j+gap] = t;
                    }else{
                        break;
                    }
                }
            }
        }
        gap = gap/2;
    }
}
```

```
void change(long *a)
{
    long tmp[10];
    int i;
    for(i=0; i<10; i++){
        tmp[i] = a[i];
    }
    for(i=0; i<10; i++){
        a[i] = tmp[9 - i];
    }
}
```

Web site and support <website and support>

Web site for Renesas Technology

http://japan.renesas.com/

Contact information

http://japan.renesas.com/inquiry

csc@renesas.com

Revision history<revision history,rh>

Rev.	Date issued	Contents changed	
		Page	Details
1.00	2009.10.1	--	Initial edition