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.

(Note)
URL described on the body was changed to the following.
http://tool-support.renesas.com/eng/toolnews/shc/shcv7/dr_shv7.html

date: 2002/4/15

HITACHI SEMICONDUCTOR TECHNICAL UPDATE

Classification of Production	Development Environment			No	TN-CSX-036A/E
THEME	SuperH RISC engine C/C++ Compiler Ver.7.0.04 bug report	Classification of Information	1. Spec change 2. Supplement of Documents (3.) Limitation of Use	4. Change of Mask 5. Change of Production Line	
PRODUCT NAME	SH-1,SH-2,SH-2E, SH2-DSP,SH-3, SH3-DSP,SH-4	Lot No.: All	Reference Documents: —	Rev.: 1	Effective Date: Eternity

Attached is the description of the known bugs in Ver. 7.0.04 of the SuperH RISC engine C/C++ compiler.
Inform the customers who have the package version in the table below of the bugs.

	Package version	Compiler version
P0700CAS7-MWR	7.0B	7.0B
	7.0.01	7.0.03
	7.0.02	7.0.04
P0700CAS7-SLR	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
P0700CAS7-H7R	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04

The checker of the bugs is on the URL below for downloading.

http://www.hitachisemiconductor.com/sic/jsp/japan/eng/products/mpumcu/tool/download/caution7002.html

Attached: P0700CAS7-020405E
SuperH RISC engine C/C++ Compiler Ver. 7.0.04
Known bugs in this release

SuperH RISC engine C/C++ compiler Ver.7.0.04
Known bugs in this release

The known bugs in this release of the compiler are described below. Those bugs in the program can be found using the checker on the URL below.

http://www.hitachisemiconductor.com/sic/jsp/japan/eng/products/mpumcu/tool/download/caution7002.html

1. Illegal destruction of the R0 register.

When a parameter is passed via the stack, the R0 register may be illegally overwritten

[Example]

```
short func1(short a0, int *a1, int a2, short a3, short a4, short a5, short a6, int a7, int a8, int a9);

void func0(short a0, int *a1, int a2, short a3, short a4, short a5, short a6)
{
                    :
 r1=func1(0,a1,0,0,0,0,0,0,0,0);
 if((r1>0)&&(r1!=1)) {
   func1(a0,a1,0,a3,a4,a5,a6,0,0,0);   /* R0 is destroyed illegally. */
 }
                   :
 }
                       :
 MOV.L      R0,@(32,R15)    ; Stores R0 at @ (32, R15).
 MOV        R8,R5
 MOV        #66,R0    ; Destroys R0.
 MOV.W      @(R0,R15),R3
 MOV        R9,R6
 MOV        #70,R0    ; Destroys R0.
 MOV.W      @(R0,R15),R1
 MOV        R0,R4     ; @(32,R15) has been illegally replaced with R0.
 MOV.L      R3,@(4,R15)
 MOV.L      R1,@(8,R15)
```

```
   MOV.L      R9,@(12,R15)
   MOV.L      R9,@(16,R15)
   BSR        _func1
   MOV.L      R9,@(20,R15)
```

[Condition]

This problem may occur when both of the following conditions are satisfied

(1) The optimize=1 option is specified.

(2) A function receives a formal parameter passed via the stack.

[How to avoid the bug]

The bug can be avoided with either method of the following.

(1) Specify the optimize=0 option.

(2) Modify the source program as shown in the example below.

[Example]

```
#include <machine.h> /* for nop() */

void    func0(short a0,int *a1,int a2,short a3,short a4,short a5,short a6)
{
  int   r0;
  short r1;

    /* Copy each formal parameter passed via the stack to a local variable at the head of
      the function.    */
  short tmp4=a4,tmp5=a5,tmp6=a6;

  /* Insert nop( ) right after the code above */
  nop();

    /* Use those local variables instead of the formal parameters passed via the stack */

  if((a0<0)||(a0==1)) {
    return;
  }
  for(r0=0;r0<a2;r0++) {
  }
  for(;r0>0;) {
```

```
        r0--;
        a3=(short)a2;
        r1=func1(0,a1,0,0,0,0,0,0,0,0);
        if((r1>0)&&(r1!=1)) {
          func1(a0,a1,0,a3,tmp4,tmp5,tmp6,0,0,0); /* Change a4-a6 to tmp4-tmp6 */
        }
        if(r1==1) {
          func1(0,0,0,0,tmp4,0,0,0,0,0);          /* Change a4 to tmp4 */
          break;
        }
      }
      return;
}
```

2. Illegal branch target of the BRA instruction.

In a program having an unconditional branch, the forward branch target of the BRA instruction may be illegal if the distance from the instruction to the target is 4094 bytes.

[Condition]

This problem may occur when both of the following conditions are satisfied

(1) Either the code=machinecode option is specified, or the code option is not specified.

(2) The distance from the BRA instruction to the forward branch target is 4094 bytes.

[How to avoid the bug]

The bug can be avoided with either method of the following.

(1) Check the source code using the checker and recompile the source program having this bug with the code=asmcode option.

(2) Modify the function detected by the checker. Inserting nop( ) to change the branch distance may be effective to avoid the bug.

[Example]

```
void    func(int a) {
   if (a) {
      … /* if the condition is false, BRA with forward distance of 4094 is generated. */
   }
}
```

Change the program as follows.

```
#include <machine.h> /* for nop() */
void    func(int a) {
   if (a) {
      …
      nop( );    /* Insert nop( ) */
   }
}
```