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: Mar.18.2004

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-065A/EA	Rev.	1.0
Title	H8S, H8/300 Series C/C++ Compiler Ver.6.0.00 bug information		Information Category	Usage Limitation		
Applicable Product	PS008CAS6-MWR PS008CAS6-SLR PS008CAS6-H7R	Lot No. Ver.6.0.00	Reference Document	H8S, H8/300 Series C/C++ Compiler Assembler Optimizing Linkage Editor User's Manual REJ10B0058-0100H Rev.1.0		

Attached is the description of the detected bug information in Ver.6.0.00 of the H8S, H8/300 Series C/C++ Compiler.
The bug will affect this package version.

Attached: PS008CAS6-040217E

H8S, H8/300 Series C/C++ Compiler Ver. 6.0.00 The details of the detected bug information

Problems in the H8S, H8/300 Series C/C++ Compiler Ver. 6.0.00

Problems found in the H8S, H8/300 series C/C++ compiler Ver.6.0.00 are listed below.

1) Incorrect object by division of an unsigned variable by a constant

An incorrect code might be generated when an unsigned int (unsigned short) or an unsigned long type variable was divided by a constant value that was the n-th power of two, while the cpuexpand option was specified.

[Example]

```
unsigned int  a;
unsigned long b;
void sub(void)
{
    a = b / 2048;                    /* Incorrect code was generated due to division by the lower two bytes*/
}
```

<Incorrect>

```
_sub:
        MOV.W   @_b+2:32,R0
        SHLR.W  #11:5,R0
        MOV.W   R0,@_a:32
        RTS
```

<Correct>

```
_sub:
        MOV.L   @_b:32,ER0
        SHLR.L  #11:5,ER0
        MOV.W   R0,@_a:32
        RTS
```

[Conditions]

This problem might occur when all of the conditions in a) or b) were fulfilled.

a)

i. H8SXN, H8SXM, H8SXA, or H8SXX was specified as a cpu kind.
ii. The cpuexpand option was specified.
iii. An (unsigned char)(variable/constant) expression was described.
iv. The variable was an unsigned int or unsigned short type variable and the constant value was the n-th power of two in the range of 0 to 255.

b)

i. H8SXN, H8SXM, H8SXA, or H8SXX was specified as a cpu kind.
ii. The cpuexpand option was specified.
iii. An (unsigned int)(variable/constant) or (unsigned short)(variable/constant) expression was described.
iv. The variable was an unsigned long type and the constant value was the n-th power of two in the range of 0 to 65535.

[Solutions]

Take either of the following methods to prevent this problem.

a) Assign the result of the operation to a variable of the same type as the dividend and then assign this variable to the left-hand side variable.

```
unsigned int  a;
unsigned long b;
void sub(void)
{
    unsigned long c;

    c = b / 2048;
    a = c;
}
```

b) Specify a divisor that was a variable with volatile and use this variable for division.

2) Incorrect structure with the initial value

The initial values were not specified to the members of the structure, when the members were pointer type and consecutively scalar type structure members, and the values specified for the pointer type and scalar type members were "address + offset" and a constant, respectively.

[Example]

```
unsigned char data[2] = { 0x00, 0x11 };
const struct st_sample {
  void  *d1;
  unsigned long d3;
}st1 = { (void *)(data+1),
         0x22222222                    /* This description was not output to an object. */
       };
```

[Conditions]

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

a) H8SXN, H8SXM, H8SXA, or H8SXX was specified as a cpu kind.
b) Pointer type and scalar type members were consecutively declared in a structure.
c) The scalar type member in b) was the same size as the pointer type member.
d) The initial value of the pointer type variable in b) was represented as "address + constant".
e) The initial value of the scalar type member in b) was a constant.
f) The const qualifier was specified for the structure.

[Solution]

This problem could be prevented by the following method:
Insert a dummy member of a different size between those members mentioned in b).
Example:

```
unsigned char data[2] = { 0x00, 0x11 };
const struct st_sample {
  void  *d1;
  unsigned int u;
  unsigned long d3;
}st1 = { (void *)(data+1),
         0xff,
         0x22222222
       };
```

3) Incorrect bit field setting

An incorrect mask value might be set when a constant value was set to a bit field with the specific bit width and bit offset.

[Example]

```
struct ST{
  signed long offset:3;
  signed long data:20;
}st;

void func(void)
{
  st.data = 0x000000ff;              /* When a constant value was assigned to, the mask value is incorrect. */
}
```

<Incorrect>

```
_func:
      MOV.L   @_st:32,ER0
      AND.L   #h'dfffb3df:32,ER0
      OR.L    #h'0001fe00:32,ER0
      MOV.L   ER0,@_st:32
      RTS
```

<Correct>

```
_func:
        MOV.L   @_st:32,ER0
        AND.L   #h'e00001ff:32,ER0
        OR.L    #h'0001fe00:32,ER0
        MOV.L   ER0,@_st:32
        RTS
```

[Conditions]

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

a) H8SXN, H8SXM, H8SXA, or H8SXX was specified as a cpu kind.
b) A constant value was set to the bit field.
c) The bit field had one of the following specifications:

i.	type: long	bit width: 20	bit offset: 3
ii.	type: long	bit width: 15	bit offset: 5
iii.	type: long	bit width: 9	bit offset: 17
iv.	type: long	bit width: 9	bit offset: 8
v.	type: long	bit width: 9	bit offset: 0
vi.	type: int /short	bit width: 9	bit offset: 2

[Solutions]

This problem could be prevented by the following method:
Insert a dummy bit field before the said bit field so that condition c) would not be fulfilled.

<Example>

```
struct ST{
  signed long offset:3;
  signed long dummy:1;              /* Insert a dummy variable */
  signed long data:20;
}st;
```

4) Incorrect value setting with using a 3-byte structure

When structreg option was specified, an incorrect value was set to the next area of the 3-byte structure when the return value was the structure, or when a value was set to the 3-byte structure of the array or pointer type.

[Example]

```
#pragma pack 1
typedef struct {
  char a;
  int b;
} ST;
#pragma unpack

ST st2[3];
ST sub();

void main(void){
  st2[1] = sub();                   /* An incorrect value was set to the following area (st2[2].a) */
}
```

[Conditions]

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

a) H8SXN, H8SXM, H8SXA, or H8SXX was specified as a cpu kind.
b) The structreg option was specified.
c) A 3-byte structure was specified for the return value of a function, or a value was set to the 3-byte structure of the array or pointer type.

[Solutions]

Take either of the following methods to prevent this problem.

a) Do not specify the structreg option for compilation.
b) Add a 1-byte dummy member to the 3-byte structure (4 bytes in total).
c) Specify a volatile qualifier to the 3-byte structure.

5) Incorrect elimination of a function call

A function call might be eliminated when this function call was the end of the function and located next to the conditional statement.

[Example]

```
extern unsigned char a,b,array1[],array2[];
void func01();
void func02();
void sub(void)
{
  if(a==1) {
    if(array2[1]==0x01) {
      func02( ); /* The function call func02 was eliminated because there was no processing after calling func02 within this function.*/
    } else {
      func01( );
    }
  } else if(a==2) {
    if(b&0x01) {
      func02( );
    } else {
      func01( );
    }
  } else {
    if(b&0x01) {
      array1[1]=0x08;
    } else {
      func01( );
    }
  }
}
 void func01(){}
 void func02(){}
```

[Conditions]

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

a) 300,300HN, 300HA, 2000N, 2000A, 2600N or 2600A was specified as a cpu kind
b) Register save/restore operation was not performed at the entry/exit of the caller function.
c) The caller function had a conditional statement and the next statement of then clause was a function call.
d) The callee function fulfilled the following conditions:
   i. No stack was used to pass the parameter to the callee function.
   ii. The return value was 4 bytes or less (when 300 was specified as the cpu kind, the return value is 2 bytes or less).
e) The callee function was the last execution in the caller function.
f) The callee function and the caller function were defined in the same file.

[Solutions]

Take either of the following methods to prevent this problem.

a) Specify the goptimize option for compilation.
b) Move the definition of the callee function to another file.

6) Incorrect result of operation on a two-dimensional array address

An incorrect result might be generated when addition or subtraction was performed on a two-dimensional array address reference expression twice or more.

[Example]

```
char *p,array[12][12];
unsigned  long x1,x2,x3;
void sub()
{
  p = (&array[x1][0] + x2 + x3);        /* Incorrectly referred to the data of the array, not the address value. */
}
```

[Conditions]
This problem might occur when all of the following conditions were fulfilled.
a) 300, 300HN, 300HA, 2000N, 2000A, 2600N, or 2600A was specified as a cpu kind.
b) The optimize=0 option was specified.
c) Addition, subtraction, or combination of the two was performed on a two-dimensional array address twice or more.
d) The two-dimensional array address was described as "&array[variable][0]".

[Solutions]
Take one of the following methods to prevent this problem.
a) Figure out the two-dimensional array address by array[variable].
[Example]
```
p = (array[x1] + x2 + x3);
```
b) Split the expression and perform addition or subtraction for each.
[Example]
```
p = (&array[x1][0] + x2);
p = (p + x3);
```
c) Specify optimization (optimize=1: default).

7) Incorrect setting and reference of bit field
An incorrect memory location might be accessed when a bit field was accessed by using the BFST/BFLD instruction.
[Example]
<Incorrect>
```
    MOV.L       #(_p+4),ER1             ; (_p+4) is set to ER1
    BFLD        #7,@ER1,R0L
    CMP.B       #4:8,R0L
    BHI         L29:8
    MOV.B       #2:8,R0L
    BRA         L38:8
L31:
                    :
L35:
    BFLD        #15,@ER3,R0L
    ADD.B       #2:8,R0L
    MOV.L       #(_p+2),ER3             ; (_p+2) is set to ER3
    MOV.L       ER3,ER2
    MOV.B       R0L,R1L
    BFST        R1L,#240,@ER2
    MOV.L       ER3,ER1                 ; ER3, in other words, (_p+2), is set to ER1
    BFLD        #240,@ER1,R2L
    MOV.B       R2L,R0H
    BFST        R0H,#15,@(_p+3):32
L29:
    MOV.B       #6:8,R0L
L38:
    BFST
                R0L,#7,@ER1         ; BFST  R0L,#7,@(_p+4) in fact, because ER1 may be changed
    SUB.B       R0L,R0L
    SUB.B       R0L,R0L
```

[Conditions]
This problem might occur when all of the following conditions were fulfilled.
a) H8SXN, H8SXM, H8SXA, or H8SXX was specified as a cpu kind.
b) The optimize=1 option (default) was specified.
c) The bit field was such that the available size was 7 bits or less and the bit offset added to the bit field size was within abyte range.
d) The same bit field was set or referred to by two or more statements in a function.
e) The same bit field was set or referred to by using the BFST/BFLD instruction.
f) The addressing mode for accessing the same bit field were different.
[Example]

```
BFST   R0L,#7,@ER1
BFST   R0L,#7,@L3+3
```

[Solution]
This problem could be prevented by the following method:
Do not specify optimize option (optimize=0).

8) Incorrect replacement of a loop control variable
An incorrect object might be generated when there was a 1-byte type variable in a loop that had a loop control variable with the type of 2 bytes or more.
[Example]

```
int a[100],b[100];
int i;
unsigned char x;
void f(void){
  x = 3;
  i = 0;
  while (i <= 32760) {              /* x was used as a loop control variable and this while statement became an endless loop. */
    a[x] = b[x];
    x++;
    i+=4;
  }
}
```

[Conditions]
This problem might occur when all of the following conditions were fulfilled.
a) H8SXN, H8SXM, H8SXA, or H8SXX was specified as a cpu kind.
b) The optimize=1 option (default) was specified.
c) A loop had a loop control variable with the type of 2 bytes or more (i in the above example).
d) A char type or unsigned char type variable (x in the above example) was described in the loop.
e) A constant value was added or subtracted to the loop control variable.
f) The variable d) was incremented.

[Solutions]
Take one of the following methods to prevent this problem.
a) Specify a value other than 1 as the value added to the variable in the loop.
b) Change the type so that the size of the variable in condition d) will be equal to or larger than that of the loop control variable.
c) Change the type so that the size of the loop control variable will be equal to or larger than that of the variable in condition d).

9) Incorrect constant propagation

A value of the then or else clause in the if statement might be incorrectly propagated immediately before an assignment expression of which the right-hand side and left-hand side were the same external variable without volatile qualifier.

[Example]

```
int x;

int sub(){
  if (x>=9999){
    x=1050;
  }
  x=x;                                  // Eliminated as an unnecessary expression
  x++;                                  // 1050 was incorrectly propagated and converted to x=1051

  return (x);
}
```

[Conditions]

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

a) H8SXN, H8SXM, H8SXA, or H8SXX was specified as a cpu kind.
b) The optimize=1 option (default) was specified.
c) There was an assignment expression of which the right-hand side and left-hand side had the same external variable without volatile qualifier.
d) There was an if statement before the assignment expression of c) and the then or else clause included an assignment to the external variable.

[Solution]

This problem could be prevented by the following method:
Specify the opt_range=noblock option.

10) Incorrect unification of string data

String data might be incorrectly unified when struct arrays or multi-dimensional arrays that had a string as a first member were defined and the same string was specified as an initial value of the first member.

[Example]

/* An initial value of a first member was string "Hello" and the size was 12 byte */

```
const char a[2][6] = {"Hello",
                      {1,2,3,4,5,6} };
const char b[2][6] = {"Hello",
                      {6,5,4,3,2,1} };

      .SECTION    C,DATA,ALIGN=2
_a:                                   ; static: a
_b:                                   ; static: b    unified incorrectly
      .SDATAZ     "Hello"
      .DATA.B     H'01,H'02,H'03,H'04,H'05,H'06
```

/* An initial value of a first member was string "Hello" and the size was 10 byte */

```
typedef struct {
    char a[6];
    long l;
} st1;
typedef struct {
    char a[6];
    char b[4];
} st2;

const st1 s1[] = {"Hello",1};
const st2 s2[] = {"Hello", {0,1,2,3}};
    .SECTION    C,DATA,ALIGN=2
_ss:                                  ; static: ss
```

```
_tt:                                    ; static: tt    unified incorrectly as ss
      .SDATAZ     "Hello"
      .DATA.L     H'00000001
```

[Conditions]

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

a) H8SXN, H8SXM, H8SXA or H8SXX was specified as the CPU type.
b) More than one struct arrays or multi-dimensional arrays that had a string as a first member were defined with const qualifier.
c) The same string was specified as a initial value of the first member of b).
d) The struct array or multi-dimensional array of b) had same data size.

[Solution]

This problem could be prevented by the following method:
Do not specify the const qualifier.

11) Cpuexpand option was not enabled

Not having generated a code for expanded interpretation when compilation was performed by specifying the cpuexpand option.
[Example] -cpu=2600a –cpuexpand were specified as compile option.

```
unsigned long ul1;
unsigned int   ui1;
void sub()
{
    ul1 = ui1 * (-1) ;
}
```

<Incorrect>

```
_func:
       MOV.W  @_ui1:32,R0
       NEG.W  R0             ; 2byte * 2byte(-1) -> 2byte
       EXTU.L ER0            ; zero-extended to 4byte
       MOV.L  ER0,@_ul1:32
```

<Correct>

```
_func:
       MOV.W      @_ui1:32,R0
       MOV.W      #-1,E0
       MULXU.W    E0,ER0          ; 2byte * 2byte -> 4byte
       MOV.L      ER0,@_ul1:32
```

[Conditions]

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

a) 300, 300HN, 300HA, 2000N, 2000A, 2600N, or 2600A was specified as a cpu kind.
b) The cpuexpand option was specified.
c) The following expressions were described:
   i. long = int(short) * (-1)
   ii. unsigned long = unsigned int(unsigned short) * (-1)

[Solution]

Take one of the following methods to prevent this problem.

a) Insert the cast to the multiplier as follows.
   When the multiplier is unsigned type, change the multiplicand from “-1” to “0xFFFF” in addition.
   Example:

```
long l1;
short s1;
unsigned long ul1;
unsigned short us1;
void sub(void){
    l1 = (long)s1 * (-1);               // It is necessary to describe “–1”, when multiplier is signed type.
    ul1 = (unsigned long)us1 * 0xFFFF; // It is necessary to describe “0xFFFF”, when multiplier is unsigned type.
}
```

b) Substitute the multiplier once to a variable with volatile qualifier, and use this variable for multiplication.
   Example:

```
long l1;
short s1;
unsigned long ul1;
```

```
unsigned short us1;
void sub(void){
    volatile short x = (-1);

    l1 = s1 * x;
    ul1 = us1 * x;
}
```

c) Do not specify the cpuexpand option.

12) Incorrect substitution with using 4 or less bytes structure

When the structure size was 4 or less bytes and the structure member was struct array, instruction of substitution the structure might not generate.

Moreover, when the structure member size was 3-byte, it might output internal error.

[Example]

```
typedef struct {
    char c1;
    char d1;
}ST1;

typedef struct {
    char d1;
    ST1 sx[1];
}ST3;

ST1 G_S;

ST3 sub()
{
     ST3 st3;
     st3.sx[0] = G_S;
     return st3;
}
```

<Incorrect>

```
_sub:
     subs    #4,sp
     ; This code was not generated
     mov.l   @(8:2,sp),er0
     mov.w   @sp,@er0
     mov.b   @(2:2,sp),@(2:2,er0)
     adds    #4,sp
     rts
```

<Correct>

```
_sub:
     push.l  er6
     mov.w   @_G_S:32,r0
     mov.l   @(8:16,sp),er1
     mov.w   r0,@er1
     mov.b   r6h,@(2:16,er1)
     pop.l   er6
     rts
```

[Conditions]

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

a) H8SXN, H8SXM, H8SXA or H8SXX was specified as the CPU type.
b) The optimize=1 option (default) was specified.
c) The size of a structure was 4-byte or less and it was declared as a local variable.
d) The structure member of c) had array of structure.
e) The structure type array of d) was assigned to a value.

[Solution]

Take one of the following methods to prevent this problem.

a) Do not specify the optimize option (optimize=0).
b) Specify a volatile qualifier to the structure of a local variable.