HITACHI SEMICONDUCTOR TECHNICAL UPDATE

Classification of Production	Development Environment			No	TN-CSX-038A/E
THEME	SuperH RISC engine C/C++ Compiler Ver.7.0 bug report (2)	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 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: P0700CAS7-020514E
SuperH RISC engine C/C++ Compiler Ver. 7
Known bugs in this release (2)

SuperH RISC engine C/C++ compiler Ver.7
Known bugs in this release (2)

The known bugs in this release of the compiler are described below. Instances of those bugs in the program except those of item 4, 7, 9, and 12 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 deletion of the save/restoration of PR register

When the following program is compiled with the speed option, the saving and restoring code of PR register may be illegally deleted.

[Example]

```
int x;
extern void f1();
extern void f2();
void f() {
    if (x == 2){
        f1();        // The then-clause ends with a function call
    }
    f2();            // The function end with a function call
    return;
}
```

```
_f:
          MOV.L       L14,R6     ; _x
          MOV.L       @R6,R0
          CMP/EQ      #2,R0
          BT          L11
L12:
          MOV.L       L14+4,R2   ; _f2
          JMP         @R2        ; The function ends with a function call,
          NOP                    ; so JSR changed into JMP and RTS is deleted
L11:
          MOV.L       L14+8,R2   ; _f1
          JSR         @R2        ; Saving and restoring code of PR register is
          NOP                    ; deleted though a function call exists
          BRA         L12
          NOP
```

[Condition]

This problem may occur when all of the following conditions are satisfied.

(1) The speed option is specified.
(2) The function ends with a function call.
(3) The if-then clause exists before the function call of (2), and the last statement of then-clause is function call.
(4) The function call is not in-line expanded.

[How to avoid the bug]

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

(1) Specify the nospeed or size option.

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

[Example]

```
#include <machine.h>        // for nop()
void f() {
    if (x == 2) {
        f1();
    }
    f2();
    nop();                  // add nop() before the return statement
    return;
}
```

2. Illegal reference of T-bit in SR register

In the following case, conditional branch may be illegally done.

(1) When following program is compiled:

[Example]

```
#include <machine.h>
extern void f();
int a;

void func() {
    int b;
    b = (a == 0);      // Sets the result of comparison to variable b
    f();               // Exists function call or set_cr()
    if (b) {           // Compares variable b with 0 or 1
        a=1;
    }
}

_func:
          STS.L       PR,@-R15
          MOV.L       L13,R6     ; _a
          MOV.L       @R6,R2
          TST         R2,R2      ; Sets the result of comparison to T-bit
          MOV.L       L13+4,R2   ; _f
          JSR         @R2        ; T-bit may be changed in the callee
          NOP                    ; function
          BF          L12        ; Refers to T-bit and branch
          MOV.L       L13,R6     ; _a
          MOV         #1,R2
          MOV.L       R2,@R6
L12:
          LDS.L       @R15+,PR
          RTS
          NOP
```

(2) When the class with destructor call which is declared in a local block in a function is written in the C++ program

[Condition]
This problem may occur when either (1) to (3) or (4) to (5) conditions are satisfied.
<for C program>
(1) The optimize=1 option is specified.
(2) A result of a comparison is set to a variable.
(3) The variable of (2) is compared with 0 or 1 after a function call or set_cr().
or
<for C++ program>
(4) The optimize=1 option is specified.
(5) The class with destructor call which is declared in a local block in a function is written in the C++ program

[How to avoid the bug]
The bug can be avoided with either method of the following.
<for C program>
(1) Specify the optimize=0 option.
(2) Qualify the variable to store the result of a comparison as volatile.
(3) Modify the source program as shown in the example below.

(a)
```
b = (a == 0) ? 1 : 0;
```

(b)
```
if (a == 0) {
        b = 1;
    } else {
        b = 0;
    }
```

<for C++ program>
Specify the optimize=0 option.

3. Illegal unification of constant values
When following program is compiled with the optimize=1 option, the constant values may be illegally unified.
[Example]
```
#define a (*(volatile unsigned short *)0x400)
#define b (*(volatile unsigned short *)0x4000)
#define c (*(volatile unsigned short *)0x402)
int d;

void func() {
    a = 0x8000;       /* (A)   */
    b = 0x8000;       /* (A')  */
    d = c + 0x8000;   /* (B)   */
}
```

```
_func:
          MOV.W       L15,R6      ; H'8000  set R6 to 0xFFFF8000
          MOV         #4,R5
          MOV         #64,R2
          SHLL8       R5
          SHLL8       R2
          MOV.W       R6,@R5      ; (A)   set variable a to 0x8000
          MOV.W       R6,@R2      ; (A')  set variable b to 0x8000
          MOV.W       @(2,R5),R0
          EXTU.W      R0,R2
          ADD         R6,R2       ; set R2 to (c+0xFFFF8000)
          MOV         R2,R6
          MOV.L       L15+4,R2    ; _d
          RTS
          MOV.L       R6,@R2      ; (B) set variable d to (c+0xFFFF8000)
                                  ;      (c+0x00008000) is correct
```

[Condition]

This problem may occur when all of the following conditions are satisfied.

(1) The optimize=1 option is specified.

(2) The same value from 128 to 255 or from 32768 to 65535 is used more than once in the function.

(3) The value of (2) is used with different sizes.

For above example, (A) and (A') are used as a 2-byte value and (B) is used as a 4-byte value.

[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]

```
int value = 0x8000;
void func() {
    a = value;        /* (A)   */
    b = value;        /* (A')  */
    d = c + value;    /* (B)   */
}
```

4. Illegal generation of a literal pool

When a program is compiled with the align16 option, the reference to a literal pool may be illegal.

When both code=machinecode and goptimize are specified, the error may occur at linkage.

When code=machinecode is specified, an illegal object code may be created at compilation.

When code=asmcode is specified, the error may occur in assembling.

[Example]

```
                :
        MOV.L       L154+2,R2  ; (1) L158  refers to literal (A)
        MOV.L       R2,@R15
        MOV.L       L154+6,R2  ; (2) _printf  refers to literal (B)
        JSR         @R2
        NOP
                :
        .ALIGN      16
L86:
        ADD         #1,R2
        BRA         L153       ; unconditional branch is created and
        MOV.L       R2,@R4     ; a literal pool is generated
L154:
        .RES.W      1
        .DATA.L     L158       ; (A) literal which cannot be reached
                               ;     from (1)
        .DATA.L     _printf    ; (B) literal which cannot be reached
                               ;     from (2)
        .ALIGN      16
L153:
                :
```

[Condition]

This problem may occur when all of the following conditions are satisfied.

(1) The align16 option is specified.

(2) An unconditional branch is created and a literal pool is generated.

[How to avoid the bug]

The bug can be avoided with the following method.

(1) Do not specify the align16 option.

5. Illegal code motion to a delay slot

When a program is compiled with the optimize=1 option, an instruction may be illegally moved to a delay slot.

[Example]

<before>

```
            :
      SHLL      R2
      MOV       R2,R0
      MOVA      L88,R0
      BRA       L144
      NOP
            :
```

<after>

```
            :
      SHLL      R2
                          ; instruction is moved to a delay slot
      MOVA      L88,R0    ; set R0
      BRA       L144
      MOV       R2,R0     ; destroy R0
```

:

[Condition]
This problem may occur when the following condition is satisfied.
(1) The same register is updated consequtively by more than one instruction.

[How to avoid the bug]
The bug can be avoided with the following method.
(1) Specify the optimize=0 option.

6. Illegal offset of a GBR-relative logical operation
When a program is compiled with the optimize=1 option and the compiler creates a GBR-relative logical operation to a 1-byte struct member, the offset may be illegal.
[Example]

```
struct {
    int a;
    unsigned char b;
} ST;
char c;

void f() {
    ST.b |= 1;
    c &= 1;
}

_f:
        STC         GBR,@-R15
        MOV         #0,R0       ; H'00000000
        LDC         R0,GBR
        MOV.L       L11+2,R0    ; H'00000008+_ST  <- (ST+4) is correct
        OR.B        #1,@(R0,GBR)
        MOV.L       L11+6,R0    ; _c
        AND.B       #1,@(R0,GBR)
        RTS
        LDC         @R15+,GBR
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) Either the gbr=user option is specified and #pragma gbr_base/gbr_base1 is used, or the gbr=auto option is specified and the map option is not specified.
(3) A global struct with a 1-byte member exists in a program.
(4) This 1-byte member is not located at the top of the struct.
(5) This member is used for logical operation in a function.
(6) This member is not used except (5).
(7) A global variable except this member exists in a function.

[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]

```
struct {
    int a;
    unsigned char b;
} ST;
char c;
unsigned char temp;    // for reference

void f() {
  ST.b |= 1;
  c &= 1;
  temp = ST.b;         // adds a reference to ST.b
}
```

7. Illegal EXTU after SWAP instruction

When a program is compiled with the optimize=1 option and a pointer is used to store the return value of swapb, swapw, or end_cnvl intrinsic function, EXTU may be illegally created after the SWAP instruction.
[Example]

```
#include <machine.h>
unsigned short *a,*b;

void func() {
    *b=swapb(*a);
}

_func:
       MOV.L      L13+2,R2  ; _a
       MOV.L      L13+6,R5  ; _b
       MOV.L      @R2,R6
       MOV.W      @R6,R2
       SWAP.B     R2,R6
       MOV.L      @R5,R2
       EXTU.B     R6,R6    ; The result of SWAP instruction is illegally expanded

        RTS
        MOV.W      R6,@R2
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) A swapb, swapw, or end_cnvl intrinsic function is used.
(3) A pointer is used to store the return value of this intrinsic function.

[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]

```
void func() {
    unsigned short temp;
    temp=swapb(*a);
    *b=temp;
}
```

8. Illegal bitfield data

When an initial value is set to a struct with an anonymous bitfield, the initial value may be illegal.

[Example]

```
struct st {
    short a:4;
    short b;
    short :12;  // anonymous bitfield
    short c:4;
} ST={1,1,3};

_ST:
          .DATA.W     H'1000
          .DATA.W     H'0001
          .DATAB.B    1,0       ; ".DATA.W H’0003"
          .DATA.W     H'0300    ; is correct.
```

[Condition]

This problem may occur when all of the following conditions are satisfied.

(1) A struct has a bitfield, an anonymous bitfield, and a member which is not a bitfield, and they are defined in the order shown below.

```
struct A {
        :
    bitfield
        :
    member which is not bitfield
    anonymous bitfield          // (A)
    bitfield                    // (B)
        :
}
```

(2) The size of the underlying type of (A) and (B) is 2-byte or 4-byte.

(3) The bitwidth of (B) is 8-bit or more.

(4) The total bitwidth size of (A) and (B) is below.

(a) When the sizes of the underlying type of (A) and (B) is 2-byte : 16-bit or less

(b) When the sizes of the underlying type of (A) and (B) is 4-byte : 32-bit or less

(5) The struct declared with an initial value.

[How to avoid the bug]
The bug can be avoided with either method of the following.
(1) Change the anonymous bitfield into one with a name (e.g. dummy).
[Example]

```
struct st {
    short a:4;
    short b;
    short dummy:12;   // dummy
    short c:4;
} ST = {1,1,0,3};
```

9. Illegal loop expansion
When the speed option or the loop option is specified, a loop conditional expression may be illegally replaced.
[Example]

```
int a[100];
void main(int n) {
    int i;

    for (i=0; i<n; i++) {
        a[i] = 0;
    }
}
```

```
_main:
          MOV       R4,R7
          ADD       #-1,R4  ; When a value of R4 is 0x80000000, underflow occurs
                            ; and R4 will have 0x7FFFFFFF.
          MOV       R4,R6
          CMP/PL    R4      ; The result of the comparison is incorrect.
          MOV       #0,R4
          BF        L12
          ADD       #-1,R6
                :
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The speed option or the loop option is specified.
(2) A loop statement is used in a function.
(3) The loop upper bound is in the range shown below.
(a) When the loop control variable, say i, is incremented like i += step,
the value is in the range from 0x80000000 to 0x80000000+step-1
(b) When the loop control variable, say i, is decremented like i -= step,
the value is in the range from 0x7FFFFFFF to 0x7FFFFFFF-step+1
When loop upper bound is in a variable and its value is in the range above, the behavior may be incorrect.

[How to avoid the bug]

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

(1) Neither the speed option nor the loop option is specified.

(2) The noloop option is specified.

10. Illegal reference to a struct or an array parameter

When a struct, a union or an array is used in a parameter of a function and this parameter is referred to in the function, the address to refer to this parameter may be illegal.

[Example]

```
typedef  struct{
    int A[10];
    double B;
    char F[20];
} ST;
extern ST f(ST a,ST b);
ST S;
extern int X;
void func(ST a,ST b) {
    ST t;
    if (a.B!=f(S,t).B){
        X++;
    }
    if (a.B!=b.B){
        X++;
    }
}
```

```
                  :
L12:
          MOV       R15,R2
          MOV.W     L15+2,R0   ; H'014C
          ADD       R0,R2
          MOV       R15,R0
          MOV.W     L15+4,R0   ; H'0190  R0 is destroyed illegally.
          ADD       R0,R0
          MOV.L     @R2,R4
          MOV.L     @(4,R2),R7
          MOV       R0,R2
          MOV.L     @R2,R6    ; An address which b.B is not located is accessed.
                 :
```

[Condition]

This problem may occur when all of the following conditions are satisfied.

(1) A function has a struct, a union or an array parameter.

(2) This parameter is referred to in the function.

[How to avoid the bug]
The bug can be avoided with the following method.
(1) Change a parameter into pointer.
[Example]

```
void func(ST *a,ST *b) {
    ST t;
    if (a->B!=f(S,t).B){
        X++;
    }
    if (a->B!=b->B){
        X++;
    }
}
```

11. Illegal deletion of a JMP instruction
When the speed option is specified, a JMP instruction may be deleted.
[Example]

```
void f(){
    int i,j=0;
    for (i=0; i<10; i++)
        if (j%2) j++;
    sub();
}

void sub() {
    :
}

_f:
                :
L20:
          ADD       #-1,R5
          TST       R5,R5
          BF        L11
          MOV.L     L23,R2    ; _sub   These instructions are deleted.
          JMP       @R2       ;                     |
          NOP                 ;                     V
L18:
          MOV       R6,R0
          AND       #1,R0
          BRA       L16
          MOV       R0,R2
L13:
          MOV       R6,R0
          AND       #1,R0
          BRA       L14
          MOV       R0,R2
_sub:
                :
```

[Condition]
This problem may occur when all of the following conditions are satisfied.
(1) The speed option is specified.
(2) The function ends with a function call.
(3) This function call is not in-line expanded.
(4) The definition of the callee function follows that of the caller function.
(5) A loop statement or a conditional statement is used in the caller function.

[How to avoid the bug]
The bug can be avoided with either method of the following.
(1) Specify the nospeed or size option.
(2) Change the location of the function definitions so that the callee function does not follow the caller function.
(3) Add the nop intrinsic function at the tail of the caller function.
[Example]

```
#include <machine.h>     // for nop
void f(){
    int i,j=0;
    for (i=0; i<10; i++)
        if (j%2) j++;
    sub();
    nop();               // adds
}
```

12. Illegal loop expansion
When the following program is compiled with the optimize=1 option, a loop conditional expression may be illegally replaced.
[Example]

```
void f1()
{
    int i;

    for (i=-1; i<INT_MAX; i++) {
        a[i] = 0;
    }
}

_f1:
          MOV.L       L13,R2      ; _a
          MOV         #-4,R6      ; H'FFFFFFFC
          MOV         R6,R5
          MOV         #0,R4       ; H'00000000
          ADD         #-4,R2
L11:
          ADD         #4,R6
          MOV.L       R4,@R2
          CMP/GE      R5,R6       ; compare with H’FFFFFFFC
          ADD         #4,R2
          BF          L11
          RTS
          NOP
```

[Condition]

This problem may occur when all of the following conditions are satisfied.

(1) The optimize=1 option is specified.
(2) A loop statement is used in the function.
(3) The result of (loop upper bound – loop lower bound) overflows.
For example : for(i=-1; i<0x7FFFFFFF; i++)

[How to avoid the bug]

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

(1) Specify the optimize=0 option.
(2) Split a loop as shown in the example below.
[Example]

```
for (i =-1;i<0x7FFFFFFF;i++) {
    func(i);
)
          ↓
func(-1);
for(i=0;i<0x7FFFFFFF;i++) {
   func(i);
}
```