date: 2002/11/06

HITACHI SEMICONDUCTOR TECHNICAL UPDATE

Classification of Production	Development Environment		No	TN-CSX-044A/E	Rev	1
THEME	SuperH RISC engine C/C++ Compiler Ver.7 bug report (5)	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	P0700CAS7-MWR P0700CAS7-SLR P0700CAS7-H7R	Lot No. All	Reference Documents	SuperH RISC engine C/C++ Compiler Assembler Optimizing Linkage Editor User's Manual ADE-702-246A Rev.2.0	Effective Date Eternity	

Attached is the description of the known bugs in Ver. 7 series 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
	7.0.03	7.0.06
	7.1.00	7.1.00
P0700CAS7-SLR	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
	7.0.04	7.0.06
	7.1.00	7.1.00
P0700CAS7-H7R	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
	7.0.04	7.0.06
	7.1.00	7.1.00

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

http://www.hitachisemiconductor.com/sic/jsp/japan/eng/products/mpumcu/tool/download/caution7100.html

Attached: P0700CAS7-021015E
SuperH RISC engine C/C++ Compiler Ver. 7
Known bugs in this release (5)

SuperH RISC engine C/C++ Compiler ver. 7
Known Bugs in This Release (5)

The failures found in the ver. 7 series of the SuperH RISC engine C/C++ compiler are listed below. You can check if a program includes these failures by using a tool for checking. The tool can be downloaded from the following URL:

http://www.hitachisemiconductor.com/sic/jsp/japan/eng/products/mpumcu/tool/download/caution7100.html

1. Generating a zero extension illegally in a loop

[Description]
A zero extension may be illegally generated when a loop includes a subtraction i = i - v with an unsigned-type variable v and a signed-type variable i.

[Example]

```
int i=319;
unsigned char v=97;
main()
{
    int f;
    for(f=0;f<5;f++){
        i = i - v;    /* The zero-extended result of – v is used in the operation. */
    }
}
```

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

(1) The optimize = 1 option is specified.
(2) A loop includes a subtraction i = i - v with an unsigned-type variable v and a signed-type variable i.
(3) i is a four-byte variable of signed int or signed long type.
(4) v is an unsigned char/short-type variable whose size is less than four bytes.
(5) The variable v is an invariant in a loop.

[Solution]
If a relevant failure exists, prevent the problem by either of the following methods.

(1) Specify the optimize = 0 option to compile the file.
(2) Specify the same type for i and v.

2. Illegal type conversion

[Description]
A conversion of a variable to char/short type may not be performed correctly if a conversion to floating-point type is attempted immediately after the conversion to char/short type.

[Example]
[C source program]

```
unsigned short US = 256;
int I;
float F;

main(){
    char c;

    c = US;        /* The short-type variable is converted to char type */
    I = 3 & c;     /* The char-type variable is allocated to a register */
    F = c;         /* Conversion from char type to float type */
}
```

[Assembly source program]

```
_main:
      MOV.L     _US,R6
      MOV.L     _I,R5
      MOV.W     @R6,R0    /* 256 -> R0 Conversion EXTU to char type is not output*/
      LDS        R0,FPUL  /* Converted to float type with 256 remained*/
         :
```

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

(1) The SH2E or SH4 is specified as the CPU.
(2) A variable of the type greater than char/short is converted to char/short type.
(3) The value before the type conversion in (2) exceeds the range of the type after the conversion.
(4) The variable after the type conversion in (2) is converted to floating-point type.
(5) The value after the type conversion in (2) is allocated to a register.

[Solution]
If a relevant failure exists, prevent the problem by the following method.

(1) Define the char/short-type variable, which is to be converted to floating-point type, as an external symbol so that it will not be allocated to registers.

[Note]
When the tool mentioned above is used to check whether the program includes a concerned part, even the functions that do not fall under the failures may be detected.

3. Illegally moving an instruction beyond the code setting FPSCR

[Description]
If cpu = sh4 is specified for compilation, the FPU instruction may be illegally moved out of the loop, beyond the code setting FPSCR.

[Example]
[C source program]

```
double dd;
struct tag {
    short aa ;
    long bb ;
    char cc:5 ;
} str ;
main()
{
    int i;
    str.bb = 10;
    str.aa = 10;
    for(i=5;i>=0;i--){
        str.cc =str.aa++  ;
        dd = str.bb ;
    }
}
```

[Assembly source program]

```
 _main:
       MOV.L       R14,@-R15
       MOV         #10,R5      ; H'0000000A
       MOV.L       L13+2,R2    ; _dd
       LDS         R5,FPUL
       MOV.L       L13+6,R7    ; _str
       FLOAT       FPUL,DR8              ;Moved out of the loop, beyond LDS R2 FPSCR.
       ADD         #8,R2
       MOV         #8,R1       ; H'00000008
           :
 L11:
       MOV.B       @(8,R7),R0
       MOV         R0,R6
           :
       DT          R4
       OR          R1,R2
       LDS         R2,FPSCR
     ; FLOAT     FPUL,DR8                    Location before being moved
       ADD         #1,R5
           :
```

[Conditions]

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

(1) The optimize = 1 option is specified.
(2) The SH4 is specified as the CPU.
(3) The fpu option is not specified.
(4) The loop includes a double-type operation.
(5) The operation in (4) is an invariant in the loop.

[Solution]

If a relevant failure exists, prevent the problem by one of the following methods.

(1) Specify the optimize = 1 option to compile the file.
(2) Specify the fpu = single/double option to compile the file so that generation of a FPSCR switching code can be prevented.
(3) Move possible invariants out of the loop.

[Note]

When the tool mentioned above is used to check whefher the program includes a concerned part, even the functions that do not fall under the failures may be detected.

4. Illegal allocation of registers in a loop

[Description]

The content of registers that have been allocated in the innermost loop may be destroyed in that loop.

[Example]

The value in the register is destroyed because the same register is allocated to several variables.

[C source program]

```
    :
for(i1 = 0; i1 < max1 ; i1++) {
        a = b;
        for(i2 = 0; i2 < max2 ; i2++) {     /* (1) */
                c = x + b;                  /* (2) */
        }                                   /* (3) */
        ans = a + c;
}
    :
```

[Assembly source program]

```
        MOV.L   @(R0,R15),R5      ; R5 is allocated to a
       :
        MOV.L   R5,@(R0,R15)      ; Saves a
        :                         ; Loop entry (1)
        MOV.L   @(R0,R15),R5      ; Allocates R5 to c
        :
   L1:
        :                         ; Loop body (2) (No reference to a)
        MOV     Rn,R5             ; Stores the result of c = x + b to R5
        :
        BF  L1
        :                         ; Loop exit (3)
        MOV.L   @(R0,R15),R5      ; Overwrites R5, not storing c
        :
        MOV.L   R5,Rn             ; Loading the value of destroyed c
```

[Conditions]

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

(1) The optimize = 1 option is specified.

(2) The innermost loop includes a variable for which a register is allocated (c in the example).

(3) There is an expression that includes the variable mentioned in (2) outside of the loop. The register (R5 in the example) allocated to the variable in (2) is also allocated to another variable (a in the example) in that expression.

[Solution]

If a relevant failure exists, prevent the problem by the following method.

(1) Specify the optimize = 0 option to compile the file.

5. Illegal deletion of FPSCR loading

[Description]

When two codes to switch double/float are output, the second loading of FPSCR may be illegally deleted.

[Example]
[C source program]

```
double d0;
float f0, f1, f2;

main
{
    int i;
    for(i = 0; i < 100; i++){
      d0++;
      f1 = f1 + f0;
    }
}
```

[Assembly source program]

```
_main:                              ; function: main
    MOV.L       L13+2,R1    ; _d0
    MOVA        L13+6,R0
           :
L11:
    STS         FPSCR,R2
    DT          R6
    OR          R5,R2
    LDS         R2,FPSCR
    FADD        DR4,DR6         ; <-- STS FPSCR,R2  must follow this line.
    AND         R4,R2
    LDS         R2,FPSCR
    BF/S         L11
    FADD        FR9,FR8
    ADD         #8,R1
            :
```

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

(1) The optimize = 1 option is specified.
(2) The SH4 is specified as the CPU.
(3) The fpu option is not specified.
(4) double-type and float-type operations are included.

[Solution]
If a relevant failure exists, prevent the problem by either of the following methods.

(1) Specify the optimize = 0 option to compile the file.
(2) Specify the fpu = single/double option to compile the file so that generation of a FPSCR switching code can be prevented.

6. Illegal sign extension for parameters to call functions

[Description]
Required sign-extension may not be performed on a parameter for the runtime routine that does not return values.

[Example]
The value of c passed to the parameter (unsigned long int) R0 for the runtime routine (_itod_a) is not sign-extended.
[C source program]

```
double D;
int I;
unsigned short US = 256;
func2(){
    char c;
    c = US;
    I = 3 & c;
    D = c;
}
```

[Assembly source program]

```
_func2:
            STS.L     PR,@-R15
            ADD       #-12,R15
            MOV.L     L20,R6     ; _US
            MOV.L     L20+4,R5   ; _I
            MOV.W     @R6,R2
            MOV.L     L20+16,R6  ; _D
            MOV       R2,R0
            AND       #3,R0
            MOV.L     R6,@R15
            MOV.L     L20+20,R6  ; __itod_a
            MOV.L     R0,@R5     ;<-- EXTS.B  R2,R2 must follow this line.
            JSR       @R6
            MOV       R2,R0
```

[Conditions]
This problem may occur when the following condition is satisfied.

(1) Either of the runtime routines listed below is used. The parameter of the function is four bytes. char/short-type variables are passed to the parameter.
_itod_a, _utod_a

[Solution]

If a relevant failure exists, prevent the problem by the following method.

(1) Assign the char/short-type variable to an int-type variable before passing it to the parameter of a relevant runtime routine.

[Example]: When the parameter for the runtime routine (_itod_a) is char type

```
double D;
char C;
int I;
unsigned short US = 256;

func2(){
    char c;
    int temp;          /* Declaration of an int-type variable for a parameter*/

   c = US;
   I = 3 & c;
   temp = c;             /*  Assigns the char-type variable to the int-type variable*/
   D = I2;               /*  The int-type parameter is used as a parameter for _itod_a*/
}
```

[Note]

When the tool mentioned above is used to check whether the program includes a concerned part, several messages may be output even if only one concerned part has been found.

7. Illegal settings for or reference to structure members

[Description]

When a member in an element that has two-dimensional or above structure arrays is referred to via a pointer, settings for or reference to the member may not be made correctly.

[Example]

```
typedef struct {
   int aaa;
} ST;
void main()
{
    ((ST (*)[2])0x10000)[0]->aaa = 100; /* 0x10000[0][0].aaa is set incorrectly*/
}
```

[Conditions]

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

(1) There are two-dimensional or above structure arrays.

(2) A member in an element with the structure arrays mentioned in (2) is set or referred to via a pointer.

[Solution]

If a relevant failure exists, prevent the problem by the following method.

(1) Assign the address of a structure array to a pointer variable so that it will be one-dimensional for the pointer, before setting or referring to the member.

[Example]

```
typedef struct {
       int aaa;
   } ST;

   void main()
   {
     ST (*ptr)[2];               /* Declaration of a pointer variable */

     ptr = (ST (*)[2])0x10000; /* Assigns the address of a structure array to the pointer variable */
     ptr[0]->aaa = 100;        /* Sets aaa from the assigned variable*/
   }
```