date: 2003/10/30

RENESAS TECHNICAL UPDATE

Classification of Production	Development Environment		No	TN-CSX-056A/E	Rev	1
THEME	SuperH RISC engine C/C++ Compiler Ver.7 bug report (10)	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. Ver.7.x	Reference Documents	SuperH RISC engine C/C++ Compiler Assembler Optimizing Linkage Editor User's Manual ADE-702-372A Rev.2.0	term of validity Eternity	

Attached is the description of the known bugs in Ver. 7 series of the SuperH RISC engine C/C++ compiler.
The bugs will affect the package version shown in the table below.

	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
	7.1.01 7.1.02	7.1.01
	7.1.03	7.1.02
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
	7.1.01 7.1.02	7.1.01
	7.1.03	7.1.02
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
	7.1.01 7.1.02	7.1.01
	7.1.03	7.1.02

The check tool can be downloaded from the following URL.
http://www.renesas.com/eng/products/mpumcu/tool/index.html

Attached: P0700CAS7-031015E
SuperH RISC engine C/C++ Compiler Ver. 7 Known Bugs Report (10)

SuperH RISC engine C/C++ Compiler ver. 7
Known Bugs Report (10)

The failures found in the ver. 7 series of the SuperH RISC engine C/C++ compiler are listed below.
The check tool can be downloaded from the following URL:
http://www.renesas.com/eng/products/mpumcu/tool/index.html

1. Internal error at a macro calling

[Description]

When a function macro call is described over two or more lines and the '//' type comment is described from the first column in one of the lines, an internal error occurs.

[Example]

```
#define MACRO(a, b) a, b

void func() {
    MACRO(1,0
// <- The '//' type comment is described from the first column.
    );
}
```

[Conditions]

This problem occur when all of the following conditions are satisfied.
(1) The listfile option is specified.
(2) A function macro call is described over two or more lines.
(3) The '//' type comment is described from the first column within the function macro call.

[Solution]

If a relevant failure exists, prevent the problem by one of the following methods.
(1) Specify the nolistfile option.
(2) Write the '//' type comment from the second column.
(3) Change the '//' type comment into the '/* */' type.

2. Internal error specifying a qualified name of C++

[Description]

When a qualified name is specified for #pragma specification in the C++ language, an internal error may occur.
However, it may not reappear depending on the compile environment (operating system of a host computer).

[Example]

```
class cPRINT {
public:
    static void IntPrint(void);
};
#pragma interrupt(cPRINT::IntPrint) /* The name length is a multiple of four */
void cPRINT::IntPrint(){}
```

[Conditions]

This problem may occur when all of the following conditions are satisfied.
(1) The C++ language is used.
(2) A qualified name is specified for #pragma specification.
(3) The length of the name that includes limitation is a multiple of four.

[Solution]
If a relevant failure exists, prevent the problem by one of the following methods.
(1) Change the length of the qualified name so that it would not be a multiple of four.
<Example>

```
cPRINT::InterruptPrint
```

3. Illegal branch with the align16 option specification (Only ver.7.1.02)
[Description]
In the following case, a conditional branch may be illegally done or an error may occur at assembly or linkage.
(1) The align16 option is specified.
(2) The last of a function is a conditional statement.
(3) The statement has two or more paragraphs that ends with a function call.
(4) Another function continues after the function.

[Example]

```
#include <stdio.h>

void g(
    int a,
    int b,
    char *c);

void func(int x) {
    switch (x){
    case 0:
        g(0, 0, NULL); /* The last of a conditional statement is a function call */
        break;
    default:
        g(0, 1, NULL); /* The last of a conditional statement is a function call */
        break;
    }
}

main() {                /* Another function exists */
    func(0);
}
```

[Conditions]
This problem may occur when all of the following conditions are satisfied.
(1) The optimize=1 option is specified.
(2) The align16 option is specified.
(3) The last of th function is a conditional statement, and the statement has two or more paragraphs.
(4) All paragraphs mentioned in (3) end with a function call.
(5) Another function continues after the function.

[Solution]
If a relevant failure exists, prevent the problem by one of the following methods.
(1) Specify the optimize=0 option.
(2) Cancel the align16 option specification.
(3) After the last function call, add the nop() intrinsic function.
<Example>

```
#include <stdio.h>
#include <machine.h>        /* Add */

void func(int x) {
    switch (x){
    case 0:
        g(0, 0, NULL);
        break;
    default:
        g(0, 1, NULL);
        nop();              /* Add */
        break;
    }
}
```

4. Illegal generation of DT instruction
[Description]
When comparison is performed with the cpu option other than cpu=sh1 specified, a DT instruction may be generated or an ADD instruction may be moved illegally.

[Example1]

```
struct tbl {
    char bit:1;
}S,*Sp=&S;
char a,b;
func(){
    int t;
    t = S.bit == 0;
    a = t << 1;
    if (t != 0 ) b++;
}

_func:
    MOV.L       L14,R6      ; _S
    MOV.B       @R6,R0
    TST         #128,R0
    MOVT        R6
                             ; Instructions move to (A) illegally
    TST         R0,R0       ; -> TST R2,R2
    MOVT        R2
    MOV.L       L14+4,R5    ; _a
    SHLL        R2
    ADD         #-1,R6      ; (A) illegal move
    MOV.B       R2,@R5
    EXTS.B      R6,R2       ; (A) illegal move
    TST         R2,R2
    BF          L12
    MOV.L       L14+8,R6    ; _b
    MOV.B       @R6,R2
    ADD         #1,R2
    RTS
    MOV.B       R2,@R6
            :
```

[Example2]

```
    char b;
    func2(int c, char d) {
        c = d - 1;
        b = c << 1;
        if(c != 0) {
            b++;
        }
    }

    _func:
        MOV.L       L14+2,R6    ; _b
        EXTS.B      R5,R2
        SHLL        R4          ; shifts not a result of d-1 but parameter c
        DT          R2          ; performs c=d-1; and if(c!=0)
        BT/S        L12
        MOV.B       R4,@R6
        ADD         #1,R4
        RTS
        MOV.B       R4,@R6
    L12:
        RTS
        NOP
```

[Conditions]

This problem may occur when both of (1) and (2), and either (3a) or (3b) of the following conditions are satisfied.

Instances of this bug in the program can be found using the check tool.

(1) The optimize=1 option is specified.

(2) The cpu option other than cpu=sh1 is specified.

(3a) After a bitfield is compared ('==' or '!=') with 0, the result is compared again (Example1).

(3b) The result of addition is assigned to a variable of a different type (or cast to a different type) and this is used for comparison (Example2).

[Solution]

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

(1) Specify optimize=0 option.

(2) In the case of condition (3a), the result of a bitfield comparison is assigned to a variable with volatile specification and then compared again.

(3) In the case of condition (3b), the result of an addition is assigned to a variable with volatile specification.

5. Illegal EXTS/EXTU deletion after the left shift

[Description]

When an 1-byte or 2-byte variable is used as a source variable of left-shift operation, an EXTS/EXTU instruction may be deleted illegally.

[Example]

```
short n;
void func(short s){
    sub(s <<= n);
}

_func:
    STS.L   PR,@-R15
    MOV.L   L11,R6          ; _n
    MOV.W   @R6,R1
    MOV.L   L11+4,R6        ; __sftl
    JSR     @R6
    EXTS.W  R4,R0
    MOV.L   L11+8,R2        ; _sub
    MOV     R0,R4           ; EXTS of the operation result is not carried out
    JMP     @R2
    LDS.L   @R15+,PR
```

[Conditions]

This problem may occur when all of the following conditions are satisfied.
Instances of this bug in the program can be found using the check tool.
(1) The cpu=sh1|sh2|sh2e option is specified.
(2) A compound assignment of unsigned, signed char, or short-type variable is carried out by a left shift.
(3) The result of a left shift is assigned to a larger type variable.
(4) The result is not used after the shift.
(5) The shift count is specified by a variable.

[Solution]

If a relevant failure exists, prevent the problem by the following method.
(1) Change the compound assignment into a simple assignment.

6. Illegal comparison of a result of an unsigned subtraction

[Description]

When casting the result of subtraction of unsigned variables (or expressions) to a signed type for comparison, an unsigned comparison instruction may be generated illegally.

[Example]

```
int test(unsigned int a, unsigned int b) {
    if ((int)(a - b) < 0) {
        return 1;
    } else {
        return 0;
    }
}

_test:
    CMP/HS  R5,R4   ; (unsigned int)a < (unsigned int)b
    MOVT    R0
    RTS
    XOR     #1,R0
```

[Conditions]
This problem may occur when both of the following conditions are satisfied.
Instances of this bug in the program can be found using the check tool.
(1) The result of subtraction of unsigned variables (or expressions) are compared after casting to a signed type of the same size.
(2) The result of casting is a negative value.

[Solution]
If a relevant failure exists, prevent the problem by the following method.
(1) Compare after the result of subtraction is assigned to a local variable.

7. Illegal saving/restoring the register with #pragma entry|noregsave specification
[Description]
When #pragma entry or #pragma noregsave is specified, saving and restoring of register are deleted illegally at linkage.

[Conditions]
This problem may occur when all of the following conditions are satisfied.
(1) The code=machinecode option is specified.
(2) The goptimize option is specified.
(3) A function call is included in a function for which #pragma entry or #pragma noregsave is specified.
(4) The optimize=register option is specified at linkage.

[Solution]
If a relevant failure exists, prevent the problem by one of the following methods.
(1) Specify the code=asmcode option.
(2) Cancel specification of the goptimize option.
(3) Cancel specification of the optimize=register option at linkage.

8. Illegal constant value comparison outside the range of the bitfield
[Description]
When a signed bitfield member is compared ('==' or '!=') with a constant value over the range of the bitfield, the result of the comparison may become illegal.

[Example]
```
struct tbl {
    unsigned int ib1:1;
    int ib8:8;
} bf0 = {0,-2};

#include<stdio.h>
main() {
    if (bf0.ib8 != 254) {   /* 254 (0xFE) changes into -2 */
        printf("OK\n");
    }
}
```

[Conditions]

This problem may occur when both of (1) and (2), and one of (3a), (3b), or (3c) of the following conditions are satisfied.

Instances of this bug in the program can be found using the check tool.

(1) A signed bitfield member other than 1-bit bitfield is compared with a constant value.

(2) The constant value of (1) is outside the range of the bitfield, less than the twice of the maximum, and is larger than the twice of the minimum.
For example, in the case of the 8-bit bitfield, the range is from -255 to -129 or from 128 to 254.

(3a) The bitfield size is neither 8 nor 16 bits.

(3b) The bitfield size is 8 bits, and the bit-offset is not 8, 16, or 24 bits.

(3c) The bitfield size is 16 bits, and the bit-offset is not 16 bits.

[Solution]

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

(1) Change the constant value into a value within the range of the signed bitfield.

(2) Assign the bitfield member to an int-type global variable and compare it with the constant value.