RENESAS Tool News

RENESAS TOOL NEWS on September 1, 2009: 090901/tn1

A Note on Using the C/C++ Compiler Package for the SuperH RISC engine MCU Family V.9

Please take note of the following problem in using the C/C++ compiler package for the SuperH RISC engine MCU family V.9:

- With using postfix or multiplicative expressions consisting of a induction variable and in-loop invariant variables in a program loop (SHC-0080)

1. Product and Versions Concerned

C/C++ compiler package for the SuperH RISC engine family
V.9.00 Release 00 through V.9.03 Release 00

2. C/C++ Compiler

2.1 With using postfix or multiplicative expressions consisting of a induction variable and in-loop invariant variables in a program loop (SHC-0080)

2.1.1 Symptom

If different elements of an array having a induction variable and in-loop invariant variables only as its subscripts are referenced, or a induction variable and an in-loop invariant variable are multiplied in a program loop, the result of address calculation of the array or the result of multiplication of the variables may become incorrect. Here, a induction variable is such that it is incremented or decremented in a program loop by a constant value each time after the body of the loop is executed.

2.1.2 Conditions

This problem consists of the following two symptoms, as described in Section 2.1.1:

Symptom 1:

The result of address calculation of an array may become incorrect if different elements of the array having a induction variable and in-loop invariant variables only as its subscripts are referenced in a program loop.

Symptom 2:

The result of multiplication of a induction variable and an in-loop invariant variable may become incorrect if the multiplication is performed in a program loop.

2.1.3 Conditions of Symptom 1

Symptom 1 may appear if the following conditions are all satisfied:

(1) The optimize=1 option is selected.

(2) Two or more different elements of an array of type signed char or unsigned char are referenced in a program loop.

(3) Each of the subscripts for referencing two or more elements of the array in (2) is composed of a induction variable and an in-loop invariant variable only.
This includes such a case that the example below shows, where another variable replaces the two variables, induction and in-loop invariant.

Example:

```
------------------
 k=i+x;
 a[k]; // a[i+x]
------------------
```

(4) The type of the induction variable in (3) is any one of the following:
char, unsigned char, signed short, unsigned short, signed int, unsigned int, signed long, and unsigned long

(5) The type of the in-loop invariant variable in (3) is any one of the following:
signed char, unsigned char, signed short, and unsigned short

(6) The increments or decrements of the induction variables involved in (2) and (3) above are the same.

Related Information 1

When Symptom 1 appears, the code generated by the compiler always satisfies the four conditions, (a) through (d), listed below. So, if the conditions of Symptom 1 are satisfied, check to see whether these four conditions are met. If they are, the symptom will reappear.

(a) At least one reference to an element of the array in (2) is

made to calculate the address of the array by using the address and offset of another element of the array.

(b) The offset in (a) is a SUB instruction whose operands are the in-loop invariant variables of two elements of the array.

(c) The SUB instruction in (b) is generated before the loop in Condition (2).

(d) The result of operation by the SUB instruction in (b) takes a value that cannot be expressed with the type of the larger one of the two in-loop invariant variables involved in Related Information (b).

Here, the sizes of the types are evaluated as follows:
unsigned short > signed short > unsigned char > signed char
And, the ranges of values inexpressible by types are as follows:
signed char: smaller than -128 or greater than 127
unsigned char: negative values or greater than 255
signed short: smaller than -32,768 or greater than 32,767
unsigned short: negative values or greater than 65,535

Example:

```
-------------------------------------------
signed char a[100000];  // Condition (2)
unsigned short x = 1;   // Condition (5)
unsigned short y = 5;   // Condition (5)

f()
{
   int i;             // Condition (4)
   for (i=0;i<10;i++) { // Condition (6)
      a[i+y] = a[i+x]; // Conditions (2) and (3)
   }
}
-------------------------------------------
```

Result of compilation:

```
-------------------------------------------

_f:
    MOV.L    L13+2,R1      ; _x
    MOV.L    L13+6,R2      ; _y
    MOV.W    @R1,R4        ; x
    MOV.W    @R2,R1        ; y
```

```
        MOV.L   L13+10,R6       ; _a
        SUB     R1,R4           ; Related Info (b),
                                (c), (d) (R4=x-y)
        EXTU.W  R1,R1
        ADD     R1,R6           ; R6=&a[i+y]
        EXTU.W  R4,R4           ; 2-byte 0-expansion
                                changes result of
                                  operation by SUB
                                instruction
        MOV     #10,R5          ; H'0000000A
L11:
        MOV     R4,R0           ; R0 = (unsigned
                                short)(x-y)
        MOV.B   @(R0,R6),R2     ; Related Info (a)
                                (corresponding to
                                  a[i+x], accessed by
                                &a[i+y]+(x-y))
        ADD     #-1,R5
        TST     R5,R5
        MOV.B   R2,@R6          ; a[]
        ADD     #1,R6
        BF      L11
        RTS
        NOP
```

2.1.4 Conditions of Symptom 2

Symptom 2 may appear if the following conditions are all satisfied:

(1) The optimize=1 option is selected.

(2) In a program loop exists a multiplication of a induction variable and an in-loop invariant variable.

(3) The increment or decrement of the induction variable in (2) is the value of another in-loop invariant variable.

(4) The type of the induction variable in (2) is any of the following:

char, unsigned char, signed short, unsigned short,
signed int, unsigned int, signed long, and unsigned long

(5) The type of the in-loop invariant variable in (2) and that in (3) are any of the following:
signed char, unsigned char, signed short, and
unsigned short

Related Information 2

When Symptom 2 appears, the code generated by the compiler always satisfies the two conditions, (a) and (b), listed below. So, if the conditions of Symptom 2 are satisfied, check to see whether these two conditions are met. If they are, the symptom will reappear.

(a) A Multiply instruction whose operands are the in-loop invariant variable in Condition (2) and the one in Condition (3) is generated before the loop in Condition (2).

(b) The result of operation of the Multiply instruction in (a) takes a value that cannot be expressed with the type of the larger one of the two in-loop invariant variables in Conditions (2) and (3). (See the explanations at the last of Related Information 1, in Section 2.1.3)

Example:

```
-----------------------------------------------
C source program>
int S = 0;
char n = 64;    // Condition (5)
char m = 4;     // Condition (5)
void main(void)
{
   int i;              // Condition (4)
   for (i=0;i<128;i+=n){ // Condition (3)
      S += i*m;        // Condition (2)
   }
}
// Because looping executed twice, S would be 0*m + 64*m = 256,
// but S takes a value (char)(0*m) + (char)(64*m) = 0
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------

_main:
```

```
        MOV.L   R14,@-R15
        STS.L   MACL,@-R15
        MOV.L   L14+2,R14       ; _m
        MOV     #0,R6           ; H'00000000
        MOV.L   L14+6,R2        ; _n
        MOV     R6,R5
        MOV.B   @R14,R7         ; m
        MOV.L   L14+10,R14      ; _S
        MOV.B   @R2,R1          ; n
        MULS.W  R7,R1           ; Related Info (a)
                                  and (b)
        MOV     #-128,R7        ; H'FFFFFF80
        STS     MACL,R4
        MOV.L   @R14,R2         ; S
        BRA     L11
        EXTU.B  R7,R7
L12:
        EXTS.B  R4,R4           ; Because n*m is 1-
                                  byte signed-
                                  expanded,
                                  R4=(char)(64*4)=0
        ADD     R5,R2           ; R5(=i*m) added to
                                  R2(=S). Because
                                  R5=0, R2 remains
                                  as before (=0)
        ADD     R4,R5           ; R4(=n*m) added to
                                  R5(=i*m). Because
                                  R4=0, R5 remains
                                  as before (=0).
        ADD     R1,R6
L11:
        CMP/GE  R7,R6
        BF      L12
```

```
        MOV.L   R2,@R14         ; S
        LDS.L   @R15+,MACL
        RTS
        MOV.L   @R15+,R14
```

2.1.5 Workaround

To avoid this problem, use either of the following ways:

(1) Select option optimize=0 or optimize=debug_only instead of optimize=1.

(2) Volatile-qualify either the induction variable or any one of the in-loop invariant variables.

3. Schedule of fixing the problem

We plan to fix this problem in the release of the C/C++ compiler package for the SuperH RISC engine family V.9.03 Release 01.
