RENESAS Tool News

RENESAS TOOL NEWS on June 16, 2005: RSO-SHC-050616D

Notes on Using the C/C++ Compiler Package V.9 for the SuperH RISC Engine Family of MCUs

Please take note of the eight problems described below in using the C/C++ compiler package V.9 for the SuperH RISC engine family of MCUs.

---

1. **Versions Concerned**
   C/C++ compiler package V.9.00 Release 00 through V.9.00 Release 02 for the SuperH RISC engine family

2. **Problems**

   2.1 On Calling a Function That Takes an Argument Passed via the Stack (SHC-0030)

   When a function that takes an argument passed via the stack is called, an incorrect value may be passed to the function if the function takes an address within the stack area as its argument.

   Conditions:
   This problem may occur if the following conditions are all satisfied:
   1. The optimize=1 option is selected.
   2. A function that takes an argument passed via the stack is called.
   3. The value of the argument in Condition 2 is that of an address within the stack area such as the address of a local variable.
   4. The value of the address in Condition 3 is less than 128 plus the value pointed to by the stack pointer immediately after saving any register at the beginning of the function.
   5. No register is assigned to the value of the address in Condition 3; a code for calculating it is generated immediately before the function call.

   Example of source file:

```
-----------------------------------------------
int sub(int*, int, int, int, int);
int sub2(int, int, int*, int, int);
int data1[10],data2[10],data3[10],data4[10];
```

```
void func(int c) {
   int i;
   int a[1]={0},b[10];
   for (i = 0 ; i &lt; 10 ; i++) {
      sub(b,2,3,4,5);
      sub(b,2,3,4,5);
      if (c) {
         sub2(1,2,b,4,5);
      }
      data1[i] = data2[i] + data3[i] + data4[i];
      data2[i] = data3[i] + data4[i];
   }
}
```

Result of compilation:

```
_func:
        :
    MOV    #5,R2
    MOV.L  R2,@-R15
    MOV    R15,R4
    MOV    #4,R7
    MOV    #3,R6
    MOV    #2,R5
    JSR    @R8
    ADD    #4,R4   ; Address of b, R15+8, interpreted as R15+4
        :
```

Workaround:

This problem can be circumvented in either of the following ways:

1. Use the optimize=0 option.
2. Assign the value of the address in Condition 3 to a variable qualified to be volatile at the beginning of the calling function; then use this variable as the argument passed to the function to be called.

2.2 On a Function That Contains Objects of Type double and float (SHC-0031)

In a function that contains objects of type double and float, the lower 32 bits of double or float type objects may be rewritten to others. Conditions:

This problem may occur if the following conditions are all satisfied:

1. Option cpu=sh4, cpu=sh4a, or cpu=sh2afpu is selected.
2. Neither option fpu=single nor option fpu=double is selected.

3. A variable or constant of type float is defined or used in a function.
4. The function in Condition 3 satisfies any of the following conditions:
   4a. It takes an argument of type double.
   4b. It calls a function that takes an argument of type double or has a return value.
   4c. A type conversion between double and unsigned long long or long long is made in the function.
5. Any of these registers, FR1, FR3, FR5, FR7, FR9, FR11, is assigned to the variable in Condition 3.

Example of source file:

```
----------------------------------------
float f1,f2,f3,f4;
double d4;
void func(double d1, double d2, double d3) {
   f1 += f1;
   d4 = d3;
   f4 = f1 + f3;
   f4 += f4;
}
----------------------------------------------
 Result of compilation:
----------------------------------------------
_func:                  ; DR8 assigned to d3
     MOV.L   L11+2,R1      ; _f1
     MOV.L   L11+6,R4      ; H'00000008+_d4
     FMOV.S  @R1,FR9       ; f1 assigned to FR9 (lower 32 bits of d3)
     MOV.L   L11+10,R5     ; _f3
     FADD    FR9,FR9
     MOV.L   L11+14,R6     ; _f4
     FMOV.S  FR9,@-R4      ; Value of d3 assigned to d4
     FMOV.S  FR8,@-R4
     FMOV.S  @R5,FR8
     FMOV.S  FR9,@R1
     FADD    FR8,FR9
     FMOV.S  FR9,FR8
     FADD    FR9,FR8
     RTS

     FMOV.S  FR8,@R6
----------------------------------------
```

Workaround:

This problem can be circumvented in any of the following ways:

1. Use the fpu=single or fpu=double option.
2. If Condition 4a is satisfied, qualify the argument to be volatile.
3. If Condition 4b is satisfied, assign the argument or the return value to a variable qualified to be volatile; then use it.
4. If Condition 4c is satisfied, assign the variable or constant of type double to a variable qualified to be volatile; then use it.

2.3 On the Return Value of a Function That Calls a strcpy() Function or Contains an Assignment Expression to a Member of a Packed Structure (SHC-0032)

In a function that calls a strcpy() function or contains an assignment expression to a member of a packed structure, a return value from the function may become incorrect.

Conditions:

This problem may occur if the following conditions are all satisfied:

1. The cpu=sh2a or cpu=sh2afpu option is selected.
2. The optimize=1 option is selected.
3. A function exists whose return value is the integer or pointer type (including the float type if option cpu=sh2a used).
4. Immediately before the return statement of the function in Condition 3, a call to a strcpy() function or an assignment to a member of a packed structure is made.
5. The return value from the function in Condition 3 does not depend on the call or assignment.

Example of source file:

```
----------------------------------------
#include <string.h>
char *a,*b;
int func() {
   strcpy(a,b);
   return (0);
}
----------------------------------------------
 Result of compilation:
----------------------------------------------
_func:
     MOV.L   L11+2,R6      ; _a
     MOV.L   L11+6,R7      ; _b
     MOV.L   @R6,R0
     MOV     #0,R2         ; Return value set to R2
     MOV.L   L11+10,R4     ; __slow_strcpy
```

```
    JMP     @R4          ; JSR replaced with JMP
    MOV.L   @R7,R1
                         ; Saving/restoring PR; and RTV/N R2
                                                removed
----------------------------------------
```

Workaround:
This problem can be circumvented in any of the following ways:

1. Within the function satisfying the conditions concerned, define a dummy volatile variable (only to define it is effective).
2. Place an nop() include function immediately before the return statement.
3. Use the optimize=0 option.

2.4 On Performing a Product-Sum Operation Containing a Pointer-Type Variable in a loop (SHC-0033)

If a product-sum operation containing a pointer-type variable is performed in the loop of a for, while or do-while statement, a MAC.W instruction may incorrectly be generated.

Conditions:
This problem may occur if the following conditions are all satisfied:

1. The optimize=1 option is selected.
2. None of the options cpu=sh1, cpu=sh2a, and cpu=sh2afpu is selected.
3. In the program exists a for, while, or do-while statement.
4. In the loop in Condition 3 exists a product-sum operation (multiplication and addition may be performed in separate expressions).
5. Both the multiplier and multiplicand in Condition 4 are variables of type short, and the result of the add operation is assigned to a variable of type unsigned int or int.
6. The multiplier in Condition 4 is a variable to which a pointer- type or array-type variable is assigned.
7. In the loop in Condition 3 exists an expression where a value is assigned to the pointer-type or array-type variable in Condition 6.
8. The assignment in Condition 7 is performed prior to the assignment in Condition 6.

Example of source file:

```
----------------------------------------
#include <stdio.h>
#define MAX (10)
short A[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
short B[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
short mac2() {
```

```
    int i,sum=0;
    short *p1=A;
    short *p2=p1;
    short *q=&(B[MAX-1]);
    short *r=p1;
    short t, s;
    r++;
    sum = *q-- * *p1++;
    for(i=0; i < (MAX-1); i++){
        *p2++ = *p1;
        t = *q--;
        *r++ = 2;       // A; r and p1 point to the same address A[]
        s = *p1++;
        sum += (t * s);  // Replaced with sum += (*q++ * *p1++), and
                         // the value incremented in A above used
    }
    return (sum);
}
----------------------------------------
```

Workaround:

This problem can be circumvented in either of the following ways:

1. Use the optimize=0 option.
2. Assign the left or right term of the multiplication in Condition 4 to a variable qualified to be volatile prior to the multiplication (the variable need not be used).

2.5 On Subtracting 1 from the Variable to Which the Evaluation Result of an Equality/Inequality Expression Is Assigned (SHC-0034)

If a function call is made between the assignment of the evaluation result of an equality/inequality expression* to a variable and the subtraction of 1 from this variable, the result of subtraction may become incorrect.

* An expression performed by using any of the operators ==, !=, >, <, >=, and <=.

Conditions:

This problem may occur if the following conditions are all satisfied:

1. The optimize=1 option is selected.
2. The evaluation result of an equality/inequality expression is assigned to a variable.
3. There exists an expression subtracting 1 from the variable in Condition 2.
4. A call to a function that returns no value or returns any value except R0 is made between the assignment in Condition 2 and the subtraction in Condition 3.

Example of source file:

```
int X, Y, Z;
int A, B, C, D, E, F;
int foo() {
   return(0);
}
void bar(int a, int b, int c, int d, int e) {}
test() {
   int t, m, n, r, s, u, v, w;
   t = E;
   s = A;
   u = B;
   v = C;
   w = D;
   F = t;
   r = E;
   t = (X == 0);
   n = (Y == 0);
   m = (Z == 0);
   (void)foo();
   t = t - 1;
   n = n - 1;
   m = m - 1;
   t = -t;
   n = -n;
   m = -m;
   Y = t;
   Z = n;
   X = m;
   bar(s, u, v, w, r);
   B = s;
   C = u;
   D = v;
   E = w;
   A = r;
}
```

Result of compilation:

```
_test:
         :
```

```
    SUBC    R0,R0         ; Result of (X == 0) stored on R0
    MOV.L   L21+32,R1     ; _Z
    MOV.L   @R1,R5
    TST     R5,R5
    MOVT    R6
    MOV.L   R6,@(4,R15)
    BSR     _foo          ; Value in R0 corrupted at function called
    NOP
    MOV.L   @R15,R2
    ADD     #-1,R2
    ADD     #1,R0         ; Corrupted value in R0 referenced
    MOV.L   @(4,R15),R6
    ADD     #-1,R6
    NEG     R2,R2
    MOV     R0,R14
        :
----------------------------------------
```

Workaround:
This problem can be circumvented in any of the following ways:

1. Use the optimize=0 option.
2. Qualify the variable to be volatile to which the operational result of an equality expression is assigned.
3. Make the function in Condition 4 have a return value; then assign it to a dummy variable.

2.6 On a Result of a Residue Operation Where the Dividend and Divisor Are Constants of Powers of 2 and of Type long long (SHC-0035)

A result of a residue operation may become incorrect where the dividend and divisor are constants of powers of 2 and of type long long or unsigned long long.

Conditions:
This problem may occur if the following conditions are all satisfied:

1. In the program exists a residue operation where the divisor and the dividend are of type long long or unsigned long long.
2. The dividend in Condition 1 is either of the following:
   - 2a. A variable of type unsigned int or a member of a bit field of type unsigned int.
   - 2b. A variable of type signed int or a member of a bit field of type signed int. Here the residue operation in Condition 1 is used as the first or

the second operand of an equality operator, the other operand of whom is a constant of 0.

3. The divisor in Condition 1 is a constant of powers of 2; and is equal to or greater than 0x100000000 and equal to or less than 0x8000000000000000.

Example of source file:

```
----------------------------------------
unsigned long long X;
long long Y;
void func1(void) {
   X = X % 0x0000000800000000ull;
      // Incorrectly replaced with X & 0xFFFFFFFFFFFFFFFFull

}
void func2(void) {
   if ((Y % 0x0000000800000000ll) == 0) {
      // Incorrectly replaced with Y & 0xFFFFFFFFFFFFFFFFll
      func1();
   }
}
----------------------------------------
```

Workaround:
Assign the divisor satisfying Conditions 1 and 3 to a volatile- qualified variable, and use it as the divisor of the residue operation.

Example:

```
----------------------------------------
void func1() {
   volatile unsigned long long c=0x000000800000000ull;
   X = X % c;
}
----------------------------------------
```

2.7 On Dividing a Result of a Multiplication by an Integer Constant That is the Same as the Multiplicand or the Multiplier (SHC-0036)

If a result of a multiplication between an expression and an integer constant is divided by the same integer constant, the result of the division may become incorrect.

Conditions:
This problem may occur if the following conditions are all satisfied:

1. A multiplication is performed between an expression of type unsigned int and any integer constant except 0.

2. The result of the multiplication in Condition 1 is divided by the integer constant in Condition 1.
3. The result of the multiplication in Condition 1 exceeds the maximum value allowed to the type of the multiplication.

Example of source file:

```
----------------------------------------
unsigned int a=65536;
unsigned int b;
void func() {
  b=(65536*a)/65536;  // b=0 ((65536*65536)/65536 -> 0/65536=0) is
                      // correct, but replaced with b=a (=65536)
}
----------------------------------------
```

Workaround:
Assign the result of the multiplication satisfying the above conditions to a volatile-qualified variable, and use it as the dividend of the division operation.

Example:

```
----------------------------------------
void func() {
  volatile unsigned int t=65536*a;
  b=t/65536;
}
----------------------------------------
```

2.8 On Two Assignment Expressions to the Same Array Existing before an if Statement and in Its then Statement (SHC-0037)

When two assignment expressions to the same array exist before an if statement and in its then statement, either of these assignments may incorrectly be performed.

Conditions:
This problem may occur if the following conditions are all satisfied:

1. The optimize=1 option is selected.
2. In the program exists a loop of a for, while or do-while statement.
3. In the loop in Condition 2 exists an if statements with no else statement.
4. Before the if statement in Condition 3 and in its then statement exist assignment expressions to the same array. Hereafter the right term of the assignment expression before the if statement is called exp1, and the right term of the other exp2.
5. The types of exp1 and exp2 are different in size.
6. exp1 may takes a value outside the range allowed to the type of exp2, or

contrarily exp2 may takes a value outside the range allowed to the type of exp1.

Example of source file:

```
----------------------------------------
int A[10];
int x, y, z;
char sc0, sc1, sc2;
test() {
   int i;
   for(i = 0; i < 10; i++){
      A[i] = x + y;    // When if(z) is FALSE, value of x+y
      if(z){           // incorrectly assigned to array A[i]
         A[i] = sc0;
      }
      x++;
      y++;
      sc0++;
   }
}
----------------------------------------
```

Workaround:
This problem can be circumvented in any of the following ways:

1. Use the optimize=0 option.
2. Replace the if statement in Condition 3 with the one with an else statement; then place an nop() include function in this else statement.
3. Place an nop() include function anywhere in the then statement of the if statement in Condition 3.
4. Place an nop() include function between exp1 in Condition 4 and the if statement in Condition 3.

3. Schedule of Fixing the Problems

We plan to fix these problems at the release of the C/C++ compiler package V.9.00 Release 03 for the SuperH RISC engine family.
