[Notes]
C/C++ Compiler Package for RX Family

R20TS0073EJ0100
Rev.1.00
Sep. 01, 2016

Outline

When using the CC-RX C/C++ Compiler Package for the RX Family, take note of the problem described in this note regarding the following points.

1. Programs which include loops that should be iterated more than once (No. 39)
2. Passing an argument of a function by using the stack (No. 40)

Note: The number which follows the description of the precautionary note is an identifying number for the precaution.

1. Programs which Include Loops that should be Iterated More than Once (No. 39)

1.1 Applicable Product

CC-RX V2.00.00 to V2.05.00

1.2 Details

When certain optimization options are specified, code might be output which only loops once when the program code in the C source file is intended to loop two or more times.

1.3 Conditions

This problem may arise if the following conditions are all met:

(1) Neither the -optimize=0 nor -optimize=1 optimization option is specified.

(2) The conditional expression for the loop has the form: *formula including loop control variable* > *formula including the decisive value*.
"Formula" above also includes expressions such as the loop control variable or decisive value alone.

(3) The loop control variable in condition (2) is a variable of a signed type, and does not have the volatile attribute.

(4) The decisive value in condition (2) is neither a constant nor has the volatile attribute.

(5) The value in condition (4) is the minimum value of the variable in condition (3), and will not change during the loop.

```
1:unsigned long count;
2:signed char var1, var2;                          /* Conditions (3),(4) */
3:void func(void)
4:{
5:    count = 0x00000000;
6:    var2 = -128;                                       /* Condition (5) */
7:    for( var1 = 127; var1 > var2; var1--){count++;} /* Condition (2) */
8:    if(0x000000FF != count){funcsub();}
9:    …
10:}
```

If the 7th line following this code is for looping 255 times, and the variable “count” is set to 255, optimization will instead cause it to become a single loop.

That is, the variable “count” will become 1 on the 8th line.

1.4 Workarounds

To avoid this problem, take any of the following steps.

(1) Change the optimization option so as to not meet the condition for failure.

(2) Change the form of the loop control expression so that it does not meet the condition.

```
for( var1 = 127; var2 < var1; var1--){count++;}
```

```
for( var1 = 127; var1 >= var2+1; var1--){count++;}
```

(3) Change the loop control variable to an unsigned type of the same size or give it the volatile attribute.
In the case of changing the loop control variable to an unsigned type, both the initial value and the decisive value should be changed.

```
volatile signed char var1;
```

(4) Change the decisive value to a constant or a volatile variable.

```
volatile signed char var2;
```

1.5 Schedule for Fixing the Problem

This problem will be fixed in the next version.

2. Passing an Argument of a Function by Using the Stack (No. 40)

2.1 Applicable Product

CC-RX V2.00.00 to V2.05.00

2.2 Details

The allocation of an argument to the stack when an argument of a function is passed through the stack might be different from that described for the function call interface in the manual.

In cases where the calling function and the function called were not both compiled with either any version from among V2.01.00 and earlier versions or from among V2.02.00 and later versions, the program might not operate properly after the generated object files are combined.

2.3 Conditions

When the following conditions (1) to (4) are all met, code different from that described for the function call interface will be generated.

(1) The argument is of any of the following types and is passed in a register (referred to as argument [a] from here).

- long long, unsigned long long,
- double, long double: These only fulfill the condition when dbl_size=8 is specified.

(2) An argument passed in a general-purpose register R4 (referred to as argument [b] from here) is somewhere to the right of argument [a].

(3) An argument which is passed by using the stack (referred to as argument [c] from here) is to the right of argument [b].

(4) At least one argument which is passed by using the stack is somewhere to the left of argument [b].

The following phenomenon is seen when the above conditions (1) to (4) are met.

[V2.01.00 and earlier versions]

Of the arguments that satisfy condition (4), though all arguments of structure or union type should be allocated to addresses below that of argument [c], they are allocated to addresses above that of argument [c], and the allocation of arguments on the stack as a result of popping arguments onto the stack differs from that described for the function call interface.

[V2.02.00 and later versions]

Though all arguments that satisfy condition (4) should be allocated to addresses below that of argument [c], they are allocated to addresses above that of argument [c], and the allocation of arguments on the stack as a result of popping arguments onto the stack differs from that described for the function call interface.

The following phenomenon is seen when the below condition is also met in combination with conditions (1) to (4).

Argument [b] is of a scalar type consisting of 8 bytes, and another argument, which is passed by using the stack (referred to as argument [d] from here) is to the right of argument [c].

[V2.01.00 and earlier versions]

Of the arguments that satisfy condition (4), though all arguments of structure or union type should be allocated to addresses below those of arguments [c] and [d], they are allocated to addresses above those of arguments [c] and [d], and the allocation of arguments on the stack as a result of popping arguments onto the stack differs from that described for the function call interface.

[V2.02.00 and later versions]

Though all arguments that satisfy condition (4) should be allocated to addresses below those of arguments [c] and [d], they are allocated to addresses above those of arguments [c] and [d], and the allocation of arguments on the stack as a result of popping arguments onto the stack differs from that described for the function call interface.

2.4 Workarounds

With either workaround (1) or (2), the incorrect operation of the program can be avoided.

(1) Create all objects from which the program will be constructed with compilers which are V2.01.00 or earlier versions or V2.02.00 or later versions.*

However, this workaround does not work in cases where C/C++language functions and assembly language functions call each other and satisfy conditions (1) to (4).

*: Even though the behavior does not match the description of the function call interface, arguments will be passed normally if they are objects which were generated with V2.01.00 and earlier versions, or with V2.02.00 and later versions. The result is that the incorrect operation is automatically avoided.

(2) Change the form of the arguments so that any of the conditions (1) to (4) is not met.

Refer to the following manual for the description of the function call interface.

CC-RX Compiler User's Manual
https://www.renesas.com/search/keyword-search.html#genre=document&q=r20ut3248

2.5 Schedule for Fixing the Problem

This problem will be fixed in the next version.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep. 01, 2016	-	First edition issued

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061 Japan

Renesas Electronics Corporation

■Inquiry

http://www.renesas.com/contact/