RENESAS Tool News

RENESAS TOOL NEWS on October 1, 2015: 151001/tn2

Notes on Using the C Compiler Package for RL78 Family

When using the CC-RL C compiler package for the RL78 family, take note of the problems described in this note regarding the following points.

1. The output of code which rewrites argument values which have been pushed onto the stack (CCRL#002)
2. Return values of the memcmp, _COM_memcmp_ff, strcmp and _COM_strcmp_ff functions becoming incorrect (CCRL#003)
3. Return values of the strtoul and _COM_strtoul_ff functions becoming incorrect (CCRL#004)
4. Non-default section names being used with the reserved words __sectop and __secend, and with the startof and sizeof operators (CCRL#005)

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

---

1. The Output of Code which Rewrites Argument Values which have been Pushed onto the Stack (CCRL#002)

1.1 Applicable product and version

CC-RL V1.01.00

1.2 Description

Code output for a function call may overwrite argument values which have been pushed onto the stack.

1.3 Conditions

Such code might be produced when all of the conditions described in the following (1) to (3) are met.

(1) The -Onothing option is not designated.

(2) Any of the following conditions (2-1) to (2-3) are met.

(2-1) The size of an actual argument is 1 byte (char, structure, or

union).
(2-2) The size of an actual argument is 3 bytes (char, structure, or union).
(2-3) Two or more such arguments are referred to by far pointers.
(3) Arguments in (2) are declared with non-volatile modifiers.

Example of statements satisfying the condition when the -Onothing option is not designated

```
------------------------------------------------------------------------
void func0(void);
void func1(unsigned char c);
unsigned char data[2];
void func(unsigned char p1, unsigned char p2) {    // Conditions (2-1) &
                                                   // (3)
    func0();
    func1(data[p2]);
    data[1] = data[p2];
    data[0] = p1;
}
------------------------------------------------------------------------
```

Example of assembly code output for the above example of code that satisfies the conditions:

```
------------------------------------------------------------------------
_func:
    .STACK _func = 6
    push ax                  ; (a) Pushes p1 and p2 onto the stack
    call !!_func0
    mov a, [sp+0x00]
    shrw ax, 8+0x00000
    addw ax, #LOWW(_data)
    movw [sp+0x00], ax          ; (b) Overwrites [SP+1]
    movw de, ax
    mov a, [de]
    call !!_func1
    pop de
    push de
    mov a, [de]
    mov !LOWW(_data+0x00001), a
    mov a, [sp+0x01]            ; (c) Refers to the value
    mov !LOWW(_data), a
    pop hl
    ret
------------------------------------------------------------------------
```

(a) Pushes p1 and p2 onto the stack

The register pair AX, for which the different arguments (p1 and p2) are assigned to the higher- and lower-order bytes, are pushed onto the stack in preparation for the function.

(b) Overwrites [SP+1]
Overwrites [SP+1] when the movw instruction or push instruction writes a value to the location of [SP+0].
(c) Refers to the value
Refers to [SP+1], which has been overwritten.

1.4 Workaround
To update your program, use either of the following methods:
(1) Designate the -Onothing option.
(2) Change the type or number of arguments so that they do not satisfy condition (2).
(3) Add the volatile modifier to the arguments.

1.5 Schedule for fixing the problem
This problem will be fixed in the next version.

2. Return Values of the memcmp, _COM_memcmp_ff, strcmp and _COM_strcmp_ff Functions Becoming Incorrect (CCRL#003)
2.1 Applicable product and versions
CC-RL V1.00.00 to V1.01.00

2.2 Description
The comparison of arguments by memcmp, _COM_memcmp_ff, strcmp, _COM_strcmp_ff functions may produce an incorrect return value.

2.3 Conditions
The problem arises when condition (1) and either of conditions (2) or (3) listed below are met.

(1) Arguments are compared by any of the following functions.
- memcmp(s1, s2, n)
- COM_memcmp_ff(s1, s2, n)
- strcmp(s1, s2)
- _COM_strcmp_ff(s1, s2)

(2) The character code for s1 is 0x80 or greater, and the difference between the character codes for s1 and s2 is 0x80 or greater.
(3) The character code for s2 is 0x80 or greater, and the difference between the character codes for s1 and s2 is greater than 0x80.

Example of statements satisfying the condition:

```
----------------------------------------------------------------------
 #include
 int x1, x2, x3;
 void func(void)
 {
    x1 = strcmp("¥xc0", "¥x3e");  // Conditions (1) & (2)
                                  // The value of x1 becomes negative
                                  // rather than positive.
    x2 = strcmp("¥xc0", "¥x40");  // Conditions (1) & (2)
                                  // The value of x1 becomes negative
                                  // rather than positive.
    x3 = strcmp("¥x40", "¥xc2");  // Conditions (1) & (3)
                                  // The value of x3 becomes positive
                                  // rather than negative.
 }
----------------------------------------------------------------------
```

2.4 Workaround

There is no way to prevent this problem.

2.5 Schedule for fixing the problem

This problem will be fixed in the next version.

3. Return Values of the strtoul and _COM_strtoul_ff Functions Becoming Incorrect (CCRL#004)

3.1 Applicable product and versions

CC-RL V1.00.00 to V1.01.00

3.2 Description

Conversion of character strings to integers by the strtoul and _COM_strtoul_ff functions may produce incorrect return values.

3.3 Conditions

The problem arises when both conditions (1) and (2) listed below are met.

(1) Either of the following functions is used to compare arguments.
  - strtoul(nptr, endptr, base)
  - COM_strtoul_ff(nptr, endptr, base)

(2) nptr, the character string to be produced by conversion, has the - (minus) character as its leading character, and the value after conversion is out of the expressible range.

Example of statements satisfying the condition:

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

```
 #include
 char *endptr;
 unsigned long ans;
 void func(void)
 {
    ans = strtoul("-4294967300", &endptr, 10);  // Conditions (1) & (2)
 }
------------------------------------------------------------------------
 The value of ans becomes 1 (-ULONG_MAX) rather than ULONG_MAX.
```

3.4 Workaround

There is no way to prevent this problem.

3.5. Schedule for fixing the problem

This problem will be fixed in the next version.

4. Non-default Section Names being Used with the Reserved Words __sectop and __secend, and with the startof and sizeof Operators (CCRL#005)

4.1 Applicable product and versions

CC-RL V1.00.00 to V1.01.00

4.2 Description

When non-default section names are used for the reserved words __sectop and __secend, and for the startof and sizeof operators, the address of a label within the section becomes incorrect, and the addresses of labels and symbols allocated after that also become incorrect.

4.3 Conditions

The problem arises when all of conditions (1) to (3) listed below are met.

(1) The below reserved words or operators are used with non-default section names.
- C language statements
  Reserved word __sectop or __secend
- Assembly language statements
  startof or sizeof operator

(2) The labels are defined in the section described in (1).

(3) In the case of assembly language, the sections of names in (1) are defined on the lines below a statement in (1).

Example of statements satisfying the condition:

```
------------------------------------------------------------------------
 #pragma section bss BSEC1
 unsigned char *ucp1;  // Condition (2)
```

```
int sym1;           // Condition (2)
void func(void)
{
   ucp1 = (unsigned char *)__sectop("BSEC1_n");  // Condition (1)
}
------------------------------------------------------------------
```

Example of a linkage-map file output for the above example:

```
------------------------------------------------------------------
*** Mapping List ***

SECTION                    START     END       SIZE   ALIGN

BSEC1_n
                      000f9f0e  000f9f11       4   2

*** Symbol List ***

SECTION=BSEC1_n
FILE=DefaultBuild¥r_main.obj
                      000f9f0e  000f9f11       4
 _ucp1
                      000f9f12        2   data ,g        1
 _sym1
                      000f9f14        2   data ,g        0

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

From ucp1, the addresses of the labels and symbols are incorrect.

4.4 Workaround

Write the program and section definitions which use the reserved words __sectop and __secend, or the startof and sizeof operators, in different files.

4.5 Schedule for fixing the problem

This problem will be fixed in the next version.