[Notes]
C Compiler Package for RL78 Family

R20TS0240EJ0100
Rev.1.00
Dec. 1, 2017

Outline

When using the CC-RL C compiler package for the RL78 family, note the following point.

1. When a function has multiple arguments and also has assignment or comparison between formal arguments (CCRL#017)

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

1. When a Function Has Multiple Arguments and Also Has Assignment or Comparison between Formal Arguments (CCRL#017)

1.1 Applicable Products

CC-RL V1.02.00 to V1.05.00

1.2 Details

When a function has multiple arguments and also has assignment or comparison between formal arguments, a code that references another formal argument may be invalid.

1.3 Conditions

A code that references another formal argument may be invalid when all of conditions (1) to (2), described below, are met:

(1) The (a) or (b) processing is performed in the processing of the function after inline expansion for the function (Note).

Note: When the function contains a function call, branch, or loop, at least one of the following processing must be performed in the period from the beginning of the function to the first function call, branch, or loop. Note that this condition does not apply to a branch that substantially has only one branch destination or a loop to which loop expansion is performed by optimization.

Example of a Branch That Substantially Has Only One Branch Destination

```
if (1) {
 ...
}
else {
 ...
}
```

(a) This processing stores formal arguments passed by a register, and the storage destination address is one of the following (a-1) to (a-4).

(a-1) Address pointed by a formal argument (near pointer) passed by a register

Example

```
*r2 = r1;
```

`r1:` Formal argument passed by a register
`r2:` Formal argument (near pointer) passed by a register
Includes cases in which r1 and r2 are the same formal argument.

(a-2) Address pointed by a formal argument (near pointer) passed by a register ± a constant offset

Example

```
*(r2+1) = r1;
```

`r1:` Formal argument passed by a register
`r2:` Formal argument (near pointer) passed by a register
Includes cases in which r1 and r2 are the same formal argument.

(a-3) Address pointed by a constant (near pointer) ± a formal argument (offset) passed by a register

Example

```
*((int *)0xF8000+r2) = r1;
```

`r1:` Formal argument passed by a register
`r2:` Formal argument passed by a register
Includes cases in which r1 and r2 are the same formal argument.

(a-4) Address pointed by a formal argument (near pointer) passed by a register ± a formal argument (offset) passed by a register

Example

```
*(r2+r3) = r1;
```

`r1:` Formal argument passed by a register
`r2:` Formal argument (near pointer) passed by a register
`r3:` Formal argument passed by a register
Includes cases in which r1 and r2, or r1 and r3 are the same formal argument.

(b) This processing performs a comparison with a formal argument passed by a register, and the comparison target is one of the following (b-1) to (b-4).

(b-1) Formal argument passed by a register

Example

```
if (r1 == r2) {

    r1: Formal argument passed by a register
    r2: Formal argument passed by a register
```

(b-2) Address pointed by a formal argument (near pointer) passed by a register

Example

```
if (r1 == *r2) {

  r1: Formal argument passed by a register
  r2: Formal argument (near pointer) passed by a register
```

(b-3) Address pointed by a formal argument (near pointer) passed by a register ± a constant offset (within 8 bits)

Example

```
if (r1 == *(r2+1)) {

  r1: Formal argument passed by a register
  r2: Formal argument (near pointer) passed by a register
```

(b-4) Address pointed by a formal argument (near pointer) passed by a register ± a formal argument (offset) passed by a register

Example

```
if (r1 == *(r2+r3)) {

   r1: Formal argument passed by a register
   r2: Formal argument (near pointer) passed by a register
   r3: Formal argument passed by a register
```

(2) Data is passed into a formal argument other than the formal argument corresponding to (1) by one of the registers X, C, E, BC and DE, and the data is referenced in the function.

Example

```
i = r0;
```

`i:` Variable
`r0:` Formal argument passed by a register (X, C, E, BC or DE)

1.4 Example

The following is an example of the problem.

[C source]

```
1    typedef struct st {
2      int s1;
3      struct st* s2;
4    } ST;
5
6    void temp(void);
7
8    int func(ST* s, int r1, int r2) {
9      s->s2 = s;            // Condition (1) (a-1)
10     temp();               // First function call
11     return r1 + r2;       // Condition (2)
12   }
```

- Line 8: Formal argument s is assigned to register AX, formal argument r1 to register BC, and formal argument r2 to register DE.
- Line 9: Condition (1) (a) is met since formal argument s passed by a register is stored in member s2 of formal argument s passed by a register before the first function call in line 10.
- Line 11: Condition (2) is met since formal arguments r1 and r2 passed by a register that are not formal argument s are referenced.

[Output assembler code]

```
1   _func:
2           .STACK _func = 8
3           push bc
4           push ax
5           movw hl, ax
6           movw ax, [sp+0x00]          ;Formal argument s is referenced.
7           xchw ax, hl
8           movw [hl+0x02], ax
9           call $!_temp
10          movw ax, [sp+0x02]          ;Formal argument r1 is referenced.
11          movw bc, ax
12          movw ax, [sp+0x00]          ;Formal argument s is erroneously referenced.
13          addw ax, bc
14          addw sp, #0x04
15          ret
```

- Formal argument s is stored in stack [sp+0x00], and formal argument r1 is stored in stack [sp+0x02]. Formal argument r2 is not stored in any stack.
- Line 12: Although formal argument r2 is supposed to be referenced, formal argument s is erroneously referenced.

1.5 Workaround

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

(1) Specify the optimization option as -Onothing.

(2) Add a dummy formal argument(Note) to the function for 6 bytes from the top of the function and assign it to the register so that the original formal argument is not assigned to the register.

Note: For a structure or union member, arguments may be passed by a stack and a formal argument may not be able to be assigned to a register. Additionally, for an array variable, the size of the pointer is indicated rather than the size of the entire array variable.

[Before modification]

```
void func(int *r1, int r2, int r0)
{
  *r1 = r2;
...
```

[After modification]

```
void func(long dummy1, int dummy2, int *r1, int r2, int r0)
{
  *r1 = r2;
...
```

(3) Add a dummy call of the inline_asm specification function to the beginning of the function.

The contents of the inline_asm specification function may be empty.

[Before modification]

```
void func(int *r1, int r2, int r0)
{
  *r1 = r2;
...
```

[After modification]

```
#pragma inline_asm dummy_func
void dummy_func(void){}

void func(int *r1, int r2, int r0)
{
  dummy_func();
  *r1 = r2;
...
```

1.6 Schedule for Fixing the Problem

The problem will be fixed in CC-RL V1.06.00.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec. 1, 2017	-	First edition issued

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

Renesas Electronics Corporation

■Inquiry
https://www.renesas.com/contact/