

[Notes]

R20TS0037EJ0100

Rev.1.00

Jun. 01, 2016

RL78 Family C Compiler Package

Outline

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. Writing an instruction operand in ways which are not included in the list of instruction operations (CCRL#008)
2. Outputting code which overwrites a register for interrupt handlers (CCRL#009)

Note: The numbers which follow the descriptions of the precautionary notes are identifying numbers for the precautions.

1. Writing an Instruction Operand in Ways which are not Included in the List of Instruction Operations (CCRL#008)

1.1 Applicable Products

CC-RL V1.00.00 to V1.02.00

1.2 Details

Incorrect code may be produced when an operand of a mov instruction is written in ways other than those in the list of instruction operations by omitting the offset of the operand, since the operand is not correctly complemented in this case.

1.3 Conditions

This problem arises if any of the following conditions are met:

- (1) A mov instruction is written in the form `mov [DE],#byte` or `mov [HL],#byte`.
- (2) A mov instruction is written in the form `mov ES:[DE],#byte` or `mov ES:[HL],#byte`, and the second operand #byte is a label or a formula which includes a label.

Example:

```
mov [DE], #3
mov [HL], #sym
mov ES:[DE], #sym
```

1.4 Workaround

Write the operand without omitting the offset.

```
mov [DE+0], #3
mov [HL+0], #sym
mov ES:[DE+0], #sym
```

1.5 Schedule for Fixing the Problem

This problem will be fixed in the next version.

2. Outputting Code which Overwrites a Register for Interrupt Handlers (CCRL#009)

2.1 Applicable Products

CC-RL V1.00.00 to V1.02.00

2.2 Details

Code, which overwrites the value of an HL register which has been saved on the stack or of a register which has not been saved on the stack, may be output for interrupt handlers.

2.3 Conditions

This problem may arise if both of the following conditions are met.

However, note that code which overwrites a register which has not been saved on the stack will not be output when an interrupt handler uses fewer than 256 bytes of stack space (.STACK).

- (1) An interrupt handler is written with `#pragma interrupt` or `#pragma interrupt_brk`.
- (2) A register bank is not specified for the interrupt handler in (1).
- (3) The interrupt handler in (1) does not include a call of an `inline_asm` function.
- (4) The interrupt handler in (1) does not include a function call, or a function call is inline expanded by optimization so that the assembler code does not include a call instruction.

- Example 1

A case where the value of the HL register, which has been saved on the stack, is overwritten

```
#pragma interrupt func /* condition(1)(2) */

typedef struct
{
    unsigned int cnt;
    unsigned char flg;
} mem_t;
mem_t mem[3];

void func(void)
{
    int i;          /* Overwriting the HL register */
    for ( i=0; i<3; i++ ) {
        mem[i].cnt--;
        if ( 0 == mem[i].cnt ) {
            mem[i].flg = 1;
        }
    }
}
```

Output assembler code for the example

```

_func:
    .STACK _func = 12
    push ax
    push bc
    push de
    push hl                ; (1)
    movw ax, #LOWW(_mem)
    movw [sp+0x00], ax    ; (2) Overwriting the HL register

    pop hl                ; (1)

```

- (1) In the output assembler code, push hl is at the end of the processing to save the values of registers at the top of the interrupt handler, and pop hl is at the top of the processing to restore the value of registers at the end of the interrupt handler.
- (2) Between the push and pop instructions in (1), writing to [SP+0] proceeds while [SP+0] is holding the value of the HL register.

- Example 2

A case where a register which has not been saved on the stack is overwritten

```

#pragma interrupt func2
void func2(void) {
    volatile char arr[0x300];
    arr[2] = 1;
}

```

Output assembler code for the example

```

_func2:
    movw bc, ax    ; (1)Overwriting the BC register
    movw ax, sp
    addw ax, #0xFD00
    movw sp, ax
    movw ax, bc

```

- (1) A register which is not saved on or restored from the stack appears at the top and end of the interrupt handler. In the assembler code for the example, which is shown above, this applies to the BC register, so it is overwritten.

2.4 Workarounds

To avoid this problem, do any of the following:

- (1) Include a register bank specification in the `#pragma` directive.
- (2) Include code for a call of an empty function to be written in assembly (`#pragma inline_asm`) within the interrupt handler.
- (3) Include a function call within the interrupt handler, and prevent inline expansion by using any of `-Onothing`, `-Oinline_level=0`, or `-Oinline_level=1`.

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	Jun. 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/en-hq/support/contact.html>

Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

The past news contents have been based on information at the time of publication.

Now changed or invalid information may be included. The URLs in the Tool News also may be subject to change or become invalid without prior notice.

All trademarks and registered trademarks are the property of their respective owners.