[Note]

C Compiler Package for RH850 Family (No.30-33)

R20TS0650EJ0100
Rev.1.00
Jan. 16, 2021

Overview

When using the C compiler package for RH850 family CC-RH, note the following points.

1. Performing the tail call optimization (No.30)
2. Using the -Xintermodule option (No.31)
3. Using the -pic option (No.32)
4. Using the switch statement (No.33)

Note: The number following the note is an identification number for the precaution.

1. Performing the tail call optimization (No.30)

1.1 Applicable products

CC-RH V1.00.00 to V2.02.00

1.2 Details

Necessary type conversion may not be performed on the return value of a function.

1.3 Conditions

This problem may arise if all of the conditions from (1) to (5) are met.

(1) -Osize or -Ospeed is specified.
(2) -Otail_call=off is not specified.
(3) There is an integer-type function with a return value of either 1 byte or 2 bytes. (Note 1)
(4) There is an integer-type function whose return value type is the same size as the function (3) but with a different signedness. (Note 1)
(5) In the function of (4), the result of type conversion of the return value of the function of (3) to the return type of the function of (4) is returned.
*:Implicit type conversion is also included.

Note 1: 1- or 2-byte integer type includes the boolean type and enumerated type when -Xenum_type=auto is specified. The boolean type is regarded as a signed 1-byte type.

1.4 Examples

An example of the problem is shown below. The parts corresponding to the error conditions are shown in red.

ccrh -Osize tp.c (1) (2)

```
/* tp.c */
extern unsigned char callee();      /* (3) */
signed char caller(){               /* (4) */
  signed char returnValue;
  returnValue = callee();
  return returnValue;               /* (5) */
}
```

In this example, the return value of callee() is supposed to be sign-extended in caller() before returning, but this is not done and the upper bits are returned as 0.

1.5 Workaround

You can avoid this problem by one of the following methods. The workarounds are shown in blue.

(a) Specify the -Onothing or -Odefault option

(b) Specify the -Otail_call=off.

(c) Assign the return value of the applicable function call a volatile-qualified automatic variable before it is passed to the return statement.

```
/* tp.c */
extern unsigned char callee();      /* (3) */
signed char caller(){               /* (4) */
  volatile signed char returnValue; /* (c) */
  returnValue = callee();
  return returnValue;               /* (5) */
}
```

(d) Change the type of the return value of the caller function to a 4-byte type.

```
/* tp.c */
extern unsigned char callee();      /* (3) */
signed long caller(){               /* (d) */
  signed long returnValue;          /* (d) */
  returnValue = callee();
  return returnValue;               /* (5) */
}
```

1.6 Schedule for fixing the problem

This problem will be fixed in CC-RH V2.03.00. This version will be released in January 2021.

2. Using the -Xintermodule option (No.31)

2.1 Applicable products

CC-RH V1.00.00 to V2.02.00

2.2 Details

When the -Xintermodule option is used, access to static variables may be deleted incorrectly.

2.3 Conditions

If all of the conditions from (1) to (8) are met, access to a variable in condition (7) may be deleted incorrectly.

(1) -Xintermodule or -Xwhole_program is specified. (Note 1)

(2) -Osize or -Ospeed is specified.

(3) There is a structure-type or union-type having a pointer-type member.

(4) The pointer-type member in (3) is not const-qualified.

(5) There is a const-qualified static variable(Note 2) of the structure-type or union-type in (3).

(6) The initial value of the pointer-type member (3) of the static variable in (5) is the address of a variable.

(7) The variable with the address in (6) is a static variable(Note 2) that is not const-qualified.

(8) There is a const-qualified pointer-type static variable(Note 2) whose initial value is the address of the static variable in (5).

Note 1: When -Xwhole_program is specified, -Xintermodule is also implicitly specified.

Note 2: A static variable corresponds to a global variable or a ‘`static`’ variable.

2.4 Examples

An example of the problem is shown below. The parts corresponding to the error conditions are shown in red.

[Example]

ccrh -Osize -Xintermodule tp.c (1) (2)

```
/* tp.c */
int GGG;                    /* (7) */
typedef struct {            /* (3) */
  int* mmm;                 /* (4) */
}Str;
const Str SSS = {           /* (5) */
  &GGG                      /* (6) */
};
const Str* PPP = &SSS;      /* (8) */

int func(void) {
  GGG = 1;
  *(PPP->mmm) = 2;
  return GGG;
}
```

In this example, although function func() is supposed to return 2 because PPP->mmm points to the address of the variable GGG, it returns 1.

2.5 Workaround

You can avoid this problem by one of the following methods:

(a) Do not specify either -Xintermodule or -Xwhole_program.

(b) Specify -Odefault or -Onothing.

(c) Remove the const qualifier from the structure-type or union-type static variable in condition (5).

(d) Remove the const qualifier from the pointer-type static variable in condition (8).

2.6 Schedule for fixing the problem

This problem will be fixed in CC-RH V2.03.00. This version will be released in January 2021.

3. Using the -pic option (No.32)

3.1 Applicable products

CC-RH V1.07.00 to V2.02.00

3.2 Details

When the -pic option is used, the value of a general-purpose register r14 may be overwritten incorrectly.

3.3 Conditions

The value of a general-purpose register r14 may be overwritten incorrectly if all of the conditions from (1) to (4) are met.

(1) -pic is specified.
(2) Neither -Xswitch=ifelse nor -Xswitch=binary is specified.
(3) A switch statement is used in a function.
(4) The general-purpose register r14 is used in the output code of the function in (3).

3.4 Examples

Examples of the problem are shown below. The parts corresponding to the error conditions are shown in red.

[Example]

ccrh -pic -pirod tp.c (1) (2)

```
/* tp.c */
void fun(int x, int *y) {
  int a0 = y[0];
  int a1 = y[1];
  int a2 = y[2];
  int a3 = y[3];
  int a4 = y[4];
  int a5 = y[5];
  int a6 = y[6];
  int a7 = y[7];
  int a8 = y[8];
  switch (x) {                /* (3) */
  case 0: a0 += 1; break;
  case 1: a1 += 1; break;
  case 2: a2 += 1; break;
  case 3: a3 += 1; break;
  }
  sub(a0,a1,a2,a3,a4,a5,a6,a7,a8);
}
```

[Output code example]

```
_fun:
        .stack _fun = 24
        prepare 0x00000001, 0x00000014
        cmp 0x00000003, r6
        ld.w 0x00000020[r7], r2
        ld.w 0x0000001C[r7], r5
        ld.w 0x00000018[r7], r10
        ld.w 0x00000014[r7], r11
        ld.w 0x00000010[r7], r12
        ld.w 0x0000000C[r7], r9
        ld.w 0x00000008[r7], r8
        ld.w 0x00000004[r7], r13
        ld.w 0x00000000[r7], r14      ; (4) The r14 value specified here is、
        bh9 .BB.LABEL.1_6
.BB.LABEL.1_1:  ; entry
        shl 0x00000001, r6
        jarl .BB.LABEL.1_8, r14       ;    overwritten incorrectly here.
.BB.LABEL.1_8:
        add r6, r14
        jmp .SWITCH.LABEL.1_7-.BB.LABEL.1_8[r14]
.SWITCH.LABEL.1_7:
        br9 .BB.LABEL.1_2
        br9 .BB.LABEL.1_3
        br9 .BB.LABEL.1_4
        br9 .BB.LABEL.1_5
.SWITCH.LABEL.1_7.END:
.BB.LABEL.1_2:  ; switch_clause_bb
        add 0x00000001, r14
        br9 .BB.LABEL.1_6
.BB.LABEL.1_3:  ; switch_clause_bb30
        add 0x00000001, r13
        br9 .BB.LABEL.1_6
.BB.LABEL.1_4:  ; switch_clause_bb33
        add 0x00000001, r8
        br9 .BB.LABEL.1_6
.BB.LABEL.1_5:  ; switch_clause_bb36
        add 0x00000001, r9
.BB.LABEL.1_6:  ; switch_break_bb
        mov r13, r7
        mov r14, r6                     ; This refers to an incorrectly
overwritten value.
        st.w r2, 0x00000010[r3]
        st.w r5, 0x0000000C[r3]
        st.w r10, 0x00000008[r3]
        st.w r11, 0x00000004[r3]
        st.w r12, 0x00000000[r3]
        jarl _sub, r31
        dispose 0x00000014, 0x00000001, [r31]
```

3.5 Workaround

This problem can be avoided by specifying -Xswitch=ifelse or -Xswitch=binary.

3.6 Schedule for fixing the problem

This problem will be fixed in CC-RH V2.03.00. This version will be released in January 2021.

4. Using the switch statement (No.33)

4.1 Applicable products

CC-RH V2.02.00

4.2 Details

The value of a general-purpose register r1 may be overwritten incorrectly at runtime.

4.3 Conditions

This problem may arise if all of the conditions from (1) to (4) are met.

(1) Neither -Xswitch=ifelse nor -Xswitch=binary is specified.
(2) -Onothing is not specified.
(3) A switch statement is used in a function.
(4) One of the following conditions is met:
  (4-a) The function in (3) is subject to the function for detecting stack smashing(Note 1).
  (4-b) The stack size of the program is 4 Mbytes or more. Also the general-purpose register r1 is used to access to the stack area in the function in (3). (Note 2)

Note 1: -Xstack_protector option, -Xstack_protector_all option, or #pragma stack_protector directive is used.
Note 2: The stack size is set in the startup routine.
The general-purpose register r1 is used when accessing a location that is 4-Mbytes or more away from the stack pointer.

4.4 Examples

Examples of the problem are shown below. The parts corresponding to the error conditions are shown in red.

[Example(A)]

ccrh tp1.c (1) (2)

```
/* tp1.c */
#pragma stack_protector funA     /* (4-a) */
struct ST { char i; double d; };
int funA(struct ST x) {
  switch(x.i) {                  /* (3) */
  case 0: return x.i;
  case 1: return x.i;
  case 2: return x.i;
  case 3: return x.i;
  }
  return 0;
}
```

[Output code example(A)]

```
  _funA:
        .stack _funA = 16
        add 0xFFFFFFF0, r3
        mov 0xDEADCCCC, r1
        st.w r1, 0x0000000C[r3]
        st.w r6, 0x00000000[r3]
        st.w r7, 0x00000004[r3]
        st.w r8, 0x00000008[r3]
        ld.b 0x00000000[r3], r10
        cmp 0x00000003, r10
        bh9 .BB.LABEL.1_14
  .BB.LABEL.1_1:  ; entry
        ld.w 0x0000000C[r3], r1           ; The r1 value specified here is
        mov 0xDEADCCCC, r12
        mov r10, r1                     ; overwritten incorrectly here.
        shl 0x00000001, r1
        jmp #.SWITCH.LABEL.1_17[r1]
  .SWITCH.LABEL.1_17:
        br9 .BB.LABEL.1_2
        br9 .BB.LABEL.1_5
        br9 .BB.LABEL.1_8
        br9 .BB.LABEL.1_11
  .SWITCH.LABEL.1_17.END:
  .BB.LABEL.1_2:  ; switch_clause_bb
        cmp r12, r1                           ; This refers to an incorrectly
overwritten value.
        bnz9 .BB.LABEL.1_4
  .BB.LABEL.1_3:  ; switch_clause_bb
        dispose 0x00000010, 0x00000000, [r31]
  .BB.LABEL.1_4:  ; switch_clause_bb
        jr ___stack_chk_fail
  .BB.LABEL.1_5:  ; switch_clause_bb7
        cmp r12, r1                           ; This refers to an incorrectly
overwritten value.
        bnz9 .BB.LABEL.1_7
  .BB.LABEL.1_6:  ; switch_clause_bb7
        dispose 0x00000010, 0x00000000, [r31]
  .BB.LABEL.1_7:  ; switch_clause_bb7
        jr ___stack_chk_fail
  .BB.LABEL.1_8:  ; switch_clause_bb12
        cmp r12, r1                           ; This refers to an incorrectly
overwritten value.
        bnz9 .BB.LABEL.1_10
  .BB.LABEL.1_9:  ; switch_clause_bb12
        dispose 0x00000010, 0x00000000, [r31]
  .BB.LABEL.1_10: ; switch_clause_bb12
        jr ___stack_chk_fail
  .BB.LABEL.1_11: ; switch_clause_bb17
        cmp r12, r1                           ; This refers to an incorrectly
overwritten value.
        bnz9 .BB.LABEL.1_13
  .BB.LABEL.1_12: ; switch_clause_bb17
        dispose 0x00000010, 0x00000000, [r31]
  .BB.LABEL.1_13: ; switch_clause_bb17
        jr ___stack_chk_fail
  .BB.LABEL.1_14: ; bb23
        mov 0x00000000, r10
```

```
        ld.w 0x0000000C[r3], r1
        mov 0xDEADCCCC, r12
        cmp r12, r1
        bnz9 .BB.LABEL.1_16
  .BB.LABEL.1_15: ; bb23
        dispose 0x00000010, 0x00000000, [r31]
  .BB.LABEL.1_16: ; bb23
        jr ___stack_chk_fail
```

[Example(B)]

ccrh tp2.c (1) (2)

```
/* tp2.c */
void funB(int i) {
  volatile char d[0x400002];
  switch(i) {                /* (3) */
  case 0: d[0x400001] = 0; sub(i); break;
  case 1: d[0x400000] = 0; break;
  case 2: d[0x400000] = 0; break;
  case 3: d[0x400000] = 0; break;
  }
}
```

[Output code example(B)]

```
_funB:
        .stack _funB = 4194312                      ;(4-b)
        prepare 0x00000001, 0x0000007C
        movhi 0x0000FFC0, r3, r1
        movea 0x00000078, r1, r3
        cmp 0x00000003, r6
        bh9 .BB.LABEL.1_4
.BB.LABEL.1_1:  ; entry
        movhi 0x00000040, r3, r1          ;(4-b)The r1 value specified here is
        mov r6, r1                             ; overwritten incorrectly here.
        shl 0x00000001, r1
        jmp #.SWITCH.LABEL.1_5[r1]
.SWITCH.LABEL.1_5:
        br9 .BB.LABEL.1_2
        br9 .BB.LABEL.1_3
        br9 .BB.LABEL.1_3
        br9 .BB.LABEL.1_3
.SWITCH.LABEL.1_5.END:
.BB.LABEL.1_2:  ; switch_clause_bb
        st.b r0, 0x00000003[r1]           ;(4-b) This refers to an incorrectly
overwritten value.
        jarl _sub, r31
        br9 .BB.LABEL.1_4
.BB.LABEL.1_3:  ; switch_clause_bb11
        st.b r0, 0x00000002[r1]             ; This refers to an incorrectly
overwritten value.
.BB.LABEL.1_4:  ; return
        movhi 0x00000040, r3, r1
        movea 0xFFFFFF88, r1, r3
        dispose 0x0000007C, 0x00000001, [r31]
```

4.5 Workaround

This problem can be avoided by one of the following methods.

(a) Specify -Xswitch=ifelse or -Xswitch=binary.

(b) Specify -Onothing.

4.6 Schedule for fixing the problem

This problem will be fixed in CC-RH V2.03.00. This version will be released in January 2021.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan.16.21	-	First edition issued

Corporate Headquarters

TOYOSU FORESIA, 3- 2- 24 Toyosu,
Koto-ku, Tokyo 135- 0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

TS Colophon 4.2