# 【注意事項】
# RH850 ファミリ用 C コンパイラパッケージ
# (注意事項 No.30-33)

R20TS0650JJ0100
Rev.1.00
2021.01.16 号

## 概要

RH850 ファミリ用 C コンパイラパッケージ CC-RH の使用上の注意事項を連絡します。

1. 末尾呼び出し最適化に関する注意事項(No.30)
2. -Xintermodule オプションの使用に関する注意事項(No.31)
3. -pic オプションの使用に関する注意事項(No.32)
4. switch 文に関する注意事項(No.33)

注：注意事項の後ろの番号は、注意事項の識別番号です。

## 1. 末尾呼び出し最適化に関する注意事項(No.30)

### 1.1 該当製品

CC-RH V1.00.00~V2.02.00

### 1.2 内容

関数の戻り値に対し行うべき型変換が行われない場合があります。

### 1.3 発生条件

次の(1)から(5)のすべてを満たす場合に発生する可能性があります。

(1) -Osize または-Ospeed を指定している。
(2) -Otail_call=off を指定していない。
(3) 戻り値が 1 バイトまたは 2 バイトの整数型の関数が存在する。(注 1)
(4) 戻り値の型が(3)の関数と同じサイズで符号の有無が異なる整数型の関数が存在する。(注 1)
(5) (4)の関数内において、(3)の関数の戻り値を(4)の関数の戻り値の型に型変換*した結果を返している。
　*：暗黙の型変換を含む

注 1：1 バイトまたは 2 バイトの整数型には、ブーリアン型や-Xenum_type=auto 指定時の列挙型を含みます。ブーリアン型は符号ありの 1 バイト型とみなされます。

## 1.4 発生例

以下に発生例を記します。赤文字が発生条件の該当箇所です。

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) */
}
```

この例の場合は、callee()の戻り値を caller()内で符号拡張してから返すべきですが、
これを行わず、上位側ビットが 0 のまま返されてしまいます。

## 1.5 回避策

以下のいずれかを行うことで回避できます。青文字が回避策を実施した箇所です。

(a) -Onothing または-Odefault を指定する。

(b) -Otail_call=off を指定する。

(c) 当該関数呼び出しの戻り値を volatile 修飾した自動変数に代入してから return 文に渡す。

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

(d) 呼び出し元の関数の戻り値の型を、4 バイトの型に変換する。

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

## 1.6 恒久対策

2021 年 1 月に公開予定の CC-RH V2.03.00 で改修する予定です。

## 2. -Xintermodule オプションの使用に関する注意事項 (No.31)

### 2.1 該当製品

CC-RH V1.00.00~V2.02.00

### 2.2 内容

-Xintermodule オプションの機能を使用した場合、静的変数に対するアクセスを不正に削除する場合があります。

### 2.3 発生条件

次の(1)から(8)のすべてを満たす場合に、条件(7)の変数に対するアクセスを不正に削除する可能性があります。

(1) -Xintermodule または-Xwhole_program を指定している。(注 1)

(2) -Osize または-Ospeed を指定している。

(3) ポインタ型のメンバーを持つ構造体型または共用体型が存在する。

(4) (3)のポインタ型のメンバーは const 修飾がされていない。

(5) (3)の構造体型または共用体型の const 修飾された静的変数(注 2)が存在する。

(6) (5)の静的変数の(3)のポインタ型のメンバーの初期値は変数のアドレスである。

(7) (6)のアドレスを取られている変数は const 修飾されていない静的変数(注 2)である。

(8) (5)の静的変数のアドレスを初期値とする const 修飾されたポインタ型の静的変数(注 2)が存在する。

注 1：-Xwhole_program 指定時は-Xintermodule が暗黙に指定されます。

注 2：静的変数には大域変数と static 変数が該当します。

## 2.4 発生例

以下に発生例を記します。赤文字が発生条件の該当箇所です。

[発生例]

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;
}
```

この例の場合は、PPP->mmm は変数 GGG のアドレスを指しているため、
関数 func()は 2 を返すべきですが、1 を返してしまいます。

## 2.5 回避策

以下のいずれかを行うことで回避できます。

(a) -Xintermodule および-Xwhole_program を指定しない。

(b) -Odefault または-Onothing を指定する。

(c) 発生条件(5)の構造体型または共用体型の静的変数の const 修飾を外す。

(d) 発生条件(8)のポインタ型の静的変数の const 修飾を外す。

## 2.6 恒久対策

2021 年 1 月に公開予定の CC-RH V2.03.00 で改修する予定です。

# 3. -pic オプションの使用に関する注意事項 (No.32)

## 3.1 該当製品

CC-RH V1.07.00~V2.02.00

## 3.2 内容

-pic オプションの機能を使用した場合、汎用レジスタ r14 の値が不正に書き換えられる場合があります。

## 3.3 発生条件

次の(1)から(4)のすべてを満たす場合に、汎用レジスタ r14 の値が不正に書き換えられる可能性があります。

(1) -pic を指定している。
(2) -Xswitch=ifelse および-Xswitch=binary を指定していない。
(3) 関数内に switch 文を記述している。
(4) (3)の関数の出力コード内で汎用レジスタ r14 を使用している。

## 3.4 発生例

以下に発生例を記します。赤文字が発生条件の該当箇所です。

[発生例]

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);
}
```

[出力コード例]

```
_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) ここで設定した r14 の値が、
        bh9 .BB.LABEL.1_6
.BB.LABEL.1_1:  ; entry
        shl 0x00000001, r6
        jarl .BB.LABEL.1_8, r14       ;    ここで不正に書き換えられる
.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                     ; 不正に書き換えられた r14 の値を参照している
        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 回避策

-Xswitch=ifelse または-Xswitch=binary を指定することで回避できます。

## 3.6 恒久対策

2021 年 1 月に公開予定の CC-RH V2.03.00 で改修する予定です。

# 4. switch 文に関する注意事項(No.33)

## 4.1 該当製品

CC-RH V2.02.00

## 4.2 内容

実行時に汎用レジスタ r1 の値が不正に書き換えられる場合があります。

## 4.3 発生条件

次の(1)から(4)のすべてを満たす場合に発生する可能性があります。

(1) -Xswitch=ifelse および-Xswitch=binary を指定していない。
(2) -Onothing を指定していない。
(3) 関数内に switch 文を記述している。
(4) 次のいずれかを満たしている。
  (4-a) (3)の関数がスタック破壊検出機能(注 1)の対象である。
  (4-b) プログラムのスタックサイズが 4Mbyte より大きい。さらに(3)の関数内でスタック領域へのアクセスに汎用レジスタ r1 を使用している。(注 2)

注 1：-Xstack_protector オプション, -Xstack_protector_all オプション, #pragma stack_protector 指令を使用している。
注 2：スタックサイズは、スタートアップルーチン内で設定しています。
スタックポインタから 4Mbyte 以上離れた位置をアクセスする際に汎用レジスタ r1 を使用します。

## 4.4 発生例

以下に発生例を記します。赤文字が発生条件の該当箇所です。

[発生例(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;
}
```

[出力コード例(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          ;ここで設定した r1 の値が、
        mov 0xDEADCCCC, r12
        mov r10, r1                    ;ここで不正に書き換えられる。
        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                          ;不正に書き換えられた値を参照している
        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                          ;不正に書き換えられた値を参照している
        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                          ;不正に書き換えられた値を参照している
        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                          ;不正に書き換えられた値を参照している
        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
```

[発生例(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;
  }
}
```

[出力コード例(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)ここで設定した r1 の値が、
        mov r6, r1                                ;ここで不正に書き換えられる。
        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)不正に書き換えられた値を参照している
        jarl _sub, r31
        br9 .BB.LABEL.1_4
.BB.LABEL.1_3:  ; switch_clause_bb11
        st.b r0, 0x00000002[r1]               ;不正に書き換えられた値を参照している
.BB.LABEL.1_4:  ; return
        movhi 0x00000040, r3, r1
        movea 0xFFFFFF88, r1, r3
        dispose 0x0000007C, 0x00000001, [r31]
```

## 4.5 回避策

以下のいずれかを行うことで回避できます。

(a) -Xswitch=ifelse または-Xswitch=binary を指定する。

(b) -Onothing を指定する。

## 4.6 恒久対策

2021 年 1 月に公開予定の CC-RH V2.03.00 で改修する予定です。

以上

## 改訂記録

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 1.00 | Jan.16.21 | - | 新規発行 |
| | | | |

本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。

過去のニュース内容は発行当時の情報をもとにしており、現時点では変更された情報や無効な情報が含まれている場合があります。

ニュース本文中の URL を予告なしに変更または中止することがありますので、あらかじめご承知ください。


## 本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

## お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/



## 商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

© 2021 Renesas Electronics Corporation. All rights reserved.

TS Colophon 4.2