# 【注意事項】
# RX ファミリ用 C/C++コンパイラパッケージ
# (注意事項 No.55-58)

R20TS0649JJ0100
Rev.1.00
2021.01.16 号

## 概要

CC-RX コンパイラパッケージの使用上の注意事項を連絡します。

1. rmpab, rmpaw, rmpal または memchr を使用する場合の注意事項(No.55)
2. 末尾呼び出し最適化に関する注意事項(No.56)
3. -ip_optimize オプションの使用に関する注意事項(No.57)
4. 多次元配列を使用する場合の注意事項(No.58)

注：注意事項の後ろの番号は、注意事項の識別番号です。

## 1. rmpab, rmpaw, rmpal または memchr を使用する場合の注意事項(No.55)

### 1.1 該当製品

CC-RX V2.00.00~V3.02.00

### 1.2 内容

組み込み関数 rmpab, rmpaw, rmpal, または標準ライブラリ関数 memchr を使用するプログラムの実行結果が意図どおりにならないことがあります。

### 1.3 発生条件

次の(1)から(3)のすべてを満たす場合に発生する可能性があります。

(1) 以下のいずれかの呼び出しを行っている。
- (1-1) rmpab または__rmpab を呼び出している。
- (1-2) rmpaw または__rmpaw を呼び出している。
- (1-3) rmpal または__rmpal を呼び出している。
- (1-4) memchr を呼び出している。

(2) (1-1)から(1-3)のいずれかを満たし、コンパイルオプションに-optimize=0、-noschedule のいずれも指定していない。
(1-4)を満たし、-size 及び-avoid_cross_boundary_prefetch (注 1)を指定している。

(3) 1 つの関数内で、(1)の処理が読み出すメモリ領域(注 2)と重なる領域に、メモリ書き込みを行っている。
(インライン展開により、呼び出す側の関数内に呼び出される関数の処理が移動した場合も含みます)

注 1：V2.07.00 で追加されたオプションです。
注 2：(1-1)から(1-3)を満たす場合は第 3 または第 4 引数が指す領域、(1-4)の場合は第 1 引数が指す領域です。

## 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。

ccrx tp1.c -isa=rxv1 -optimize=2 // (2)

```
// tp1.c
signed short lhs[128];
signed short rhs[128];
long long test(void){
  long long tmp;
  lhs[0] = 0;     // (3)
  tmp  = __rmpaw(0LL, 128, lhs, rhs);   // (1-2)
  lhs[1] = 0;
  return tmp;
}
```

この例の場合は、lhs[0]への書き込みが__rmpaw の呼び出しをまたいで関数出口の方へ移動します。

その結果、__rmpaw の実行結果が意図どおりではなくなります。

## 1.5 回避策

以下のいずれかを行うことで回避できます。

(a) 発生条件(1-1)から(1-3)に該当する場合、次のいずれかを実施する。
- -noschedule を指定する。
- -optimize=0 を指定する。
- -optimize=1 を指定し、-schedule を指定しない。

(b) 発生条件(1-4)に該当する場合、-speed を指定する、または-avoid_cross_boundary_prefetch を指定しない。

## 1.6 恒久対策

2021 年 1 月に公開予定の CC-RX V3.03.00 で改修する予定です。

## 2. 末尾呼び出し最適化に関する注意事項(No.56)

### 2.1 該当製品

CC-RX V2.00.00~V3.02.00

### 2.2 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。

### 2.3 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) -optimize=0 または-optimize=1 を指定していない。
(2) 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)
(3) 戻り値の型が(2)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)
(4) (3)の関数内において、(2)の関数の戻り値を(3)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-auto_enum 指定時の列挙型や、-int_to_short 指定時の int 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。

### 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。

ccrx tp2.c -isa=rxv1 -optimize=2 // (1)

```
// tp2.c
extern unsigned char callee();    /* (2) */
signed char caller(){             /* (3) */
  signed char returnValue;
  returnValue = callee();
  return returnValue;             /* (4) */
}
```

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。

## 2.5 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) -optimize=0 または-optimize=1 を指定する。

(b) 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。

```
// tp2.c

extern unsigned char callee();      /* (2) */

signed char caller(){               /* (3) */

  volatile signed char returnValue;  /* (b) */

  returnValue = callee();

  return returnValue;                /* (4) */

}
```

(c) 呼び出し元の関数の戻り値の型を、4 バイトの型に変更する。

```
// tp2.c
extern unsigned char callee();    /* (2) */
signed long caller(){             /* (c) */
  signed long returnValue;        /* (c) */
  returnValue = callee();
  return returnValue;             /* (4) */
}
```

## 2.6 恒久対策

2021 年 1 月に公開予定の CC-RX V3.03.00 で改修する予定です。

# 3. -ip_optimize オプションの使用に関する注意事項(No.57)

## 3.1 該当製品

CC-RX V2.00.00~V3.02.00

## 3.2 内容

-ip_optimize 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。

## 3.3 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) -ip_optimize または-whole_program を指定している。(注 1)
(2) -optimize=0 または-optimize=1 を指定していない。
(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：-whole_program 指定時は-ip_optimize 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。

## 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。

ccrx -isa=rxv2 -ip_optimize 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。

## 3.5 回避策

以下のいずれかを行うことで回避できます。

(a) -ip_optimize および-whole_program を指定しない。

(b) -optimize=0 または-optimize=1 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。

## 3.6 恒久対策

2021 年 1 月に公開予定の CC-RX V3.03.00 で改修する予定です。

# 4. 多次元配列を使用する場合の注意事項(No.58)

## 4.1 該当製品

CC-RX V2.00.00-V3.02.00

## 4.2 内容

3 次元以上の多次元配列を使用するプログラムの実行結果が意図どおりにならないことがあります。

## 4.3 発生条件

次の(1)から(8)のすべてを満たす場合に、発生する可能性があります。

(1) -optimize=0 または-optimize=1 を指定していない。
(2) 3 次元以上の多次元配列が存在する。
(3) (2)の多次元配列は 1 バイトまたは 2 バイトの整数型の配列である。
(4) (2)の多次元配列は volatile 修飾も__evenaccess 修飾もされていない。
(5) (2)の多次元配列において、次のいずれかの方法で、整定数が設定されている要素が 2 つ以上ある。
  (5-a) 代入文により、整定数を代入している。
  (5-b) 多次元配列が自動変数であり、その宣言時の初期値が整定数である。
(6) 方法(5-b)で整定数を設定している場合、その初期化処理における初期値の数は多次元配列の要素数に対して不足している。(注 1)
(7) (5)の整定数が設定されている要素群に、最下位以外の添え字が異なり、かつ隣接した 2 要素が含まれる。(注 2)
(8) (7)の隣接した 2 要素に対する整定数の設定処理は同じ関数内である。

注 1：例えば以下は要素数 2×2×2=8 個に対して、初期値は 7 個しかない(不足している)ので条件を満たします。

```
signed char array[2][2][2] = {{{1, 2}, {3, 4}}, {{5, 6}, {7}}};
```

注 2：例えば、要素数が 5×5×5 の 3 次元配列 data[][][]において、以下の要素の組み合わせはいずれも「最下位以外の添え字が異なり、かつ隣接した 2 要素」という条件を満たします。
- data[0][0][4] と data[0][1][0] の組み合わせ
- data[0][4][4] と data[1][0][0] の組み合わせ
- data[3][4][4] と data[4][0][0] の組み合わせ

## 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。

ccrx -isa=rxv2 tp.c (1)

```
/* tp.c */
unsigned char aaa[2][1][3];  /* (2)(3)(4) */
unsigned char XXX, YYY, ZZZ;

void func(void) {

  if (aaa[1][0][0] == 0) {
    XXX++;
  }

  aaa[0][0][2] = 0;          /* (5-a)(7)(8) */
  aaa[1][0][0] = 100;        /* (5-a)(7)(8) */

  YYY = aaa[0][0][2];
  ZZZ = aaa[1][0][0];
}
```

この例の場合は、変数 ZZZ には配列要素 aaa[1][0][0]に代入されている値の 100 が代入されるはずですが、100 を代入する前の aaa[1][0][0]の値を代入してしまいます。

## 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) -optimize=0 または-optimize=1 を指定する。

(b) 当該多次元配列に対し、volatile 修飾あるいは__evenaccess 修飾をする。

## 4.6 恒久対策

2021 年 1 月に公開予定の CC-RX V3.03.00 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/