RENESAS ツールニュース

RENESAS TOOL NEWS 2012年03月16日：120316/tn2

RXファミリ用C/C++コンパイラパッケージ (High-performance Embedded Workshop用) ご使用上のお願い

RXファミリ用C/C++コンパイラパッケージの2件の使用上の注意事項を連絡します。

- 配列参照のアドレスを配列の型と異なるポインタにキャストする際の注意事項（RXC#021）
- 関数のすべての出口の直前に関数呼び出しがある場合の注意事項（RXC#022）

注： 各注意事項の後ろの番号は、本コンパイラパッケージの注意事項の通し番号 です。

---

1. 配列参照のアドレスを配列の型と異なるポインタにキャストする際の注意事項（RXC#021）

該当バージョン：
V.1.00 Release 00 ～ V.1.02 Release 00

内容：
配列参照のアドレスを、配列の型と異なるポインタにキャストして、その値に定数加算すると、そのアドレスからの読み出しコードが正しくありません。

発生条件：
以下をすべて満たす場合に発生することがあります。
(1) ポインタ変数、または2次元以上の配列がある。
(2) 添え字付けで参照される以下の配列の要素のアドレスを &演算子 で取得している。
　(a) (1)のポインタ変数が指す型と同じ型の配列
　(b) (1)の2次元以上の配列
(3) (2)のアドレスを(1)のポインタ変数か2次元以上の配列とは異なる型の

ポインタにキャストしている。
(4) (3)のキャスト結果に以下のいずれかを加算している。
(a) 0以外の定数
(b) 定数0以外の演算式の結果
(5) (4)の結果のアドレスを間接参照に用いてメモリを読み出している。
(6) (2)～(5)を1式で記述する。
(7) (4)(b)を満たす場合、(4)(b)の結果は0ではない。

発生例1：

```
---------------------------------------------------------------
long data01[3] = { 0x11223344, 0x55667788, 0x99aabbcc };
char a, b, c;
void func01(void)
{
   long *p = data01;         /* 条件(1)のポインタ変数 */
   a = *((char *)&p[2] + 0); /* 定数0を加算しているため、 */
                             /* (4)(a)は該当しない */
   b = *((char *)&p[2] + 1); /* 条件(2),(3),(4)(a),(5),(6) */
   c = *((char *)&p[2] + 2); /* 条件(2),(3),(4)(a),(5),(6) */
}
---------------------------------------------------------------
```

正しくは、a=0xcc, b=0xbb, c=0xaaになるが、bおよびcがいずれもaと同じ値になる。
(値はリトルエンディアンの場合)

発生例2：

```
---------------------------------------------------------------
short data02[2][3] =          /* 条件(1)の2次元以上の配列 */
{{0x1122,0x3344,0x5566},
{0x7788,0x99aa,0xbbcc}};
char a, b, c;
void func02(int x)
{
  a = *((char *)&data02[1] + 0); /* 条件(2),(3),(4)(a),(5),(6) */
  b = *((char *)&data02[1] + 2); /* 条件(2),(3),(4)(a),(5),(6) */
  c = *((char *)&data02[1] + x); /* 条件(2),(3),(4)(b),(5)～(7) */
}                                          /* (注) */
---------------------------------------------------------------
```

注: 変数xが0でない場合のみ、発生条件(7)に該当。
例えば変数xが1の場合、正しくはa=0x88, b=0xaa, c=0x77になるが、
bはaと同じ値になる。また、xが0でない場合、正しくは cはaと異なるが、
xが0以外のどの値の場合でもcはaと同じ値になる。
(値はリトルエンディアンの場合)

回避策：

以下のいずれかの方法で回避してください。
(1) 発生条件(3)のキャストした結果を、一旦変数に代入して、オフセット
加算時にその変数を使用する。
発生例1の回避例：

```
----------------------------------------------------------------
char *temp = (char *)&p[2];
*(temp + 1);
----------------------------------------------------------------
```

(2) 発生条件(4)の加算演算の + 演算子の前に "+0" を挿入する。
ただし、挿入した0と発生条件(4)の加算演算を括弧でまとめない。
発生例1の回避例：

```
----------------------------------------------------------------
*((char *)&p[2] + 0 + 1 )
----------------------------------------------------------------
```

回避策にならない例：

```
----------------------------------------------------------------
*((char *)&p[2] + (0 + 1) )
----------------------------------------------------------------
```

2. 関数のすべての出口の直前に関数呼び出しがある場合の注意事項（RXC#022）

該当バージョン：
V.1.00 Release 00 ～ V.1.02 Release 00

内容：
2つの関数AとBが続けて定義されていて、関数Aの出口の直前に関数Bか他の
関数呼び出しがある場合、関数Bの呼び出しが削除される場合があります。

発生条件：
以下の条件をすべて満たす場合に、発生する可能性があります。
(1) optimzie=2またはoptimize=maxオプションを使用している。
(2) 2つの関数AとBが連続して定義されている。
(3) 関数Aに出口が複数ある。(関数の末尾およびreturn文)
(4) 関数Aのすべての出口の直前に関数呼び出しがある。
(5) (4)の関数呼び出しの1つ以上が、関数Bの呼び出しである。

発生例：

```
----------------------------------------------------------------
void FuncB();
void FuncC(),FuncD();
long long funcLL();
void FuncA()                    /* 関数A */
{
 long long v1 = funcLL();
```

```
  if (v1) {
     FuncB();                  /* 条件(4)および(5)
                              /* このFunB()の呼び出しが削除される */
     return;                  /* 条件(3) */
  } else if (v1==1){
     FuncC();                  /* 条件(4) */
     return;                  /* 条件(3) */
  } else {
     FuncD();                  /* 条件(4) */
     return;                  /* 条件(3) */
  }
}
void FuncB(){ }                /* 条件(2) */
```

なお、該当する関数Bが複数ある場合、return直前の関数だけが削除されます。

コンパイル結果：

```
_FuncA:
        CMP        #00H,R1
        BEQ        L12
L11:
        ADD        #08H,R0
        ; <==  BRA _FuncB が削除されている。
L12:
. . . . . .
        BEQ        L15
        ADD        #08H,R0
        BRA        _FuncC
L15:
        ADD        #08H,R0
        BRA        _FuncD
_FuncB:
. . . . . .
```

回避策：

以下のいずれかの方法で回避してください。

(1) optimzie=0またはoptimize=1を使用する。

(2) 関数Aの直後に、ダミーの関数を定義する。

例: FuncBの前に、ダミーの関数Dummyを定義

回避策(2)の例：

```
void FuncB();
```

```
void FuncC(),FuncD();
long long funcLL();
void FuncA()
{
 long long v1 = funcLL();
 if (v1) {
    FuncB();
    return;
 } else if (v1==1){
    FuncC();
    return;
 } else {
    FuncD();
    return;
 }
}
void DummyFunc(){ } // ダミーの関数を定義する
void FuncB(){ }
```

---

(3) 関数Aの出口の直前にダミーの命令を挿入する
ダミーの例： 組み込み関数 nop()
回避策(3)の例：

```
#include     // for nop();
void FuncB();
void FuncC(),FuncD();
long long funcLL();
void FuncA()
{
 long long v1 = funcLL();
 if (v1) {
    FuncB();
    return;
 } else if (v1==1){
    FuncC();
    return;
 } else {
    FuncD();
    nop();   // 組み込み関数 nop() をダミーとして挿入
    return;
 }
}
void FuncB(){ }
```

---

3. 恒久対策

2件の問題はすべてV.1.02 Release 01で改修しました。
V.1.02 Release 01の詳細は、RENESAS TOOL NEWS 資料番号120316/tn3 を
参照ください。以下のURLでも参照できます。(3月21日から公開予定)
https://www.renesas.com/search/keyword-search.html#genre=document&q=120316tn3
