RENESAS ツールニュース

RENESAS TOOL NEWS 2016年2月1日：160201/tn1

RL78ファミリ および 78K0R用Cコンパイラ CA78K0R、78K0R用Cコンパイラ CC78K0R ご使用上のお願い

RL78ファミリ および 78K0R用Cコンパイラ CA78K0R、78K0R用Cコンパイラ CC78K0Rの使用上の注意事項を連絡します。

1. volatile修飾された識別子の二項演算の注意事項
2. 配列要素の前置インクリメント、前置デクリメント、後置デクリメントの 注意事項
3. near属性の配列の要素、near属性の構造体 または 共用体メンバへの farキャストでの連続アクセスの注意事項
4. strcmp関数、strncmp関数の戻り値が不正となる注意事項
5. strtoul関数の戻り値が不正となる注意事項

---

1. volatile修飾された識別子の二項演算の注意事項

1.1 該当製品 および バージョン

CA78K0R V1.70 ～ V1.71 (統合開発環境 CS+)
CA78K0R V1.20 ～ V1.70 (統合開発環境 CubeSuite+)
CA78K0R V1.00 ～ V1.10 (統合開発環境 CubeSuite)
CC78K0R V1.00 ～ V2.13 (統合開発環境 PM+)

1.2 内容

volatile修飾された識別子の二項演算結果を別の識別子に代入すると、エラーまたは 誤ったコードとなる場合があります。

1.3 発生条件

以下のすべての条件を満たす時に発生する場合があります。

(1) 代入式が以下の形のいずれかである。

```
a = b 二項演算子 c;
a *= b 二項演算子 c;
a += b 二項演算子 c;
```

a -= b 二項演算子 c;
a &= b 二項演算子 c;
a ^= b 二項演算子 c;
a |= b 二項演算子 c;

(2) (1)の二項演算子が *、+、-、<<、&、^、| のいずれかである。

(3) (1)の代入演算子の左オペランドaがchar、signed char、unsigned char型の識別子である。

(4) (1)のオペランドb および cが、(1)の左オペランドaと異なる型(注)のvolatile修飾された識別子である。

注: short、unsigned short、int、unsigned int型

発生例:

```
------------------------------------------------------------
volatile unsigned short vus1, vus2;  /* 発生条件(4) */
                                /* volatile修飾された識別子 */
unsigned char uc1;  /*発生条件(3) unsigned char型の識別子 */
void func(void)
{
   uc1 = (vus2 - vus1);            /*発生条件(1)(2) */
                                /*二項演算子"-"を使用した代入 */
}
------------------------------------------------------------
```

上記発生例の場合の出力コード例:

```
------------------------------------------------------------
; line     5 :     uc1 = (vus2 - vus1);
movw    ax,!_vus2   ; vus2の下位1バイトをxレジスタにロード
movw    ax,!_vus1   ; vus1の下位1バイトをxレジスタにロードして
                ; xレジスタの値を上書きしている
mov     a,x
sub     a,x
mov     !_uc1,a
------------------------------------------------------------
```

1.4 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(1)のオペランドb および cのvolatile修飾を外す。

回避例1:

```
------------------------------------------------------------
unsigned short vus1, vus2;    /* volatile修飾を外す */
unsigned char uc1;
void func(void)
{
   uc1 = (vus2 - vus1);
```

```
}
------------------------------------------------------------
```

(2) "b 二項演算子 c"をaの型でキャストする。

回避例2:

```
------------------------------------------------------------
volatile unsigned short vus1, vus2;
unsigned char uc1;
void func(void)
{
   uc1 = (unsigned char)(vus2 - vus1);    /* uc1の型でキャスト */
}
------------------------------------------------------------
```

2. 配列要素の前置インクリメント、前置デクリメント、後置デクリメントの注意事項

2.1 該当製品 および バージョン

CA78K0R V1.70 ～ V1.71 (統合開発環境 CS+)
CA78K0R V1.20 ～ V1.70 (統合開発環境 CubeSuite+)
CA78K0R V1.00 ～ V1.10 (統合開発環境 CubeSuite) (注)
CC78K0R V2.00 ～ V2.13 (統合開発環境 PM+) (注)
注: 後置デクリメント使用時の注意事項は、CA78K0R V1.00 ～ V1.01 および CC78K0R V1.00 ～ V2.12は対象外です。

2.2 内容

配列要素で前置インクリメント、前置デクリメント、後置デクリメントを使用すると誤ったコードになる場合があります。

2.3 発生条件

以下の発生条件1 または 発生条件2のいずれかに該当した場合に発生する場合があります。

発生条件1:
以下のすべての条件を満たす時に発生する場合があります。
(1) 関数内で、自動変数、仮引数を使っていない。
(2) 1つの式中に間接参照が2つ以上ある。
　以降、2つの間接参照を "間接参照A" および "間接参照B" と表記する。
(3) 間接参照Aは、添え字が定数でない配列要素(例: ary[idx]) または
　ポインタを使用した間接参照式(例: *ptr)である。
(4) 間接参照Bは配列要素であり、要素サイズが1バイトのfar配列で添え字が
　定数でない。
(5) 間接参照Bの配列要素を前置インクリメント または 前置デクリメントしている。

発生条件2:
以下のすべての条件を満たす時に発生する場合があります。
(1) 配列要素を参照している。
(2) 配列の要素サイズが1バイト または 2バイトである。
(3) 配列の添え字がunsigned short、unsigned int型である。
(4) 配列の添え字の値を0の状態から後置デクリメントしている。

発生条件1の発生例:

```
------------------------------------------------------------
unsigned char __far fuca1[2];
unsigned char ucaa1[2][2];
unsigned char x1;
unsigned short us1, us2;
void func(void)
{
   /* 発生条件(1)自動変数、仮引数を使っていない              */
   /* 発生条件(2)間接参照A: ucaa1[us1]、間接参照B: fuca1[us2] */
   /* 発生条件(3)添え字が定数でない配列要素: ucaa1[us1]      */
   /* 発生条件(4)要素サイズ1のfar配列で添え字が定数でない    */
   /*           配列要素: fuca1[2]                          */
   /* 発生条件(5)前置インクリメント: ++fuca1[us2]           */
   x1 = ucaa1[us1][++fuca1[us2]];
}
------------------------------------------------------------
```

上記発生例の場合の出力コード例:

```
------------------------------------------------------------
; line    13 :     x1 = ucaa1[us1][++fuca1[us2]];
movw    ax,!_us1
addw    ax,ax
addw    ax,#loww (_ucaa1)
movw    de,ax
movw    ax,!_us2
addw    ax,#loww (_fuca1)
movw    hl,ax
mov     ES,#highw (_fuca1)
inc     ES:[hl+0]
mov     ES,_@SEGHL  ; 未設定の_@SEGHLを参照している
mov     a,ES:[hl]
shrw    ax,8
addw    ax,de
movw    de,ax
mov     a,[de]
mov     !_x1,a
------------------------------------------------------------
```

発生条件2の発生例:

```
------------------------------------------------------------
unsigned char uca1[10], uc1;
unsigned short us1;
void func(void)
{
   /* 発生条件(1)配列要素を参照: uc1 = uca1[us1--]          */
   /* 発生条件(2)配列の要素サイズが1バイト: uca1[10]        */
   /* 発生条件(3)配列の添え字がunsigned short型: us1        */
   /* 発生条件(4)配列の添え字の値を0の状態から              */
   /*           後置デクリメント:  us1--                    */
   uc1 = uca1[us1--];
}
------------------------------------------------------------
```

上記発生例の場合の出力コード例:

```
------------------------------------------------------------
; line    10 :     uc1 = uca1[us1--];
decw    !_us1           ; us1が0の場合
movw    bc,!_us1        ; bcレジスタの値は0xffff
mov     a,_uca1+1[bc]   ; uca1[1+0xffff]を代入
mov     !_uc1,a
------------------------------------------------------------
```

2.4 回避策

該当する発生条件で回避策が異なります。

発生条件1の場合:

以下のいずれかの方法で回避してください。

(1) far配列をnear配列に変える。

(2) テンポラリ変数を設け、前置インクリメント または 前置デクリメントの
　演算結果をテンポラリ変数に代入しながら演算する。

発生条件1の回避例:

```
------------------------------------------------------------
unsigned char __far fuca1[2];
unsigned char ucaa1[2][2];
unsigned char x1;
unsigned short us1, us2;
void func(void)
{
   unsigned char temp;      /* テンポラリ変数 */
   temp = ++fuca1[us2];     /* 前置インクリメント結果を代入 */
   x1 = ucaa1[us1][temp];
```

```
}
------------------------------------------------------------
```

発生条件2の場合:
配列の参照と添え字のデクリメントの実行を分ける。

発生条件2の回避例:

```
------------------------------------------------------------
unsigned char uca1[10], uc1;
unsigned short us1;
void func(void)
{
   uc1 = uca1[us1];        /* 配列要素の参照 */
   us1--;                  /* デクリメント */
}
------------------------------------------------------------
```

3. near属性の配列の要素、near属性の構造体 または 共用体メンバへのfarキャストでの連続アクセスの注意事項

3.1 該当製品 および バージョン

CA78K0R V1.70 ～ V1.71 (統合開発環境 CS+)
CA78K0R V1.20 ～ V1.70 (統合開発環境 CubeSuite+)
CA78K0R V1.00 ～ V1.10 (統合開発環境 CubeSuite)
CC78K0R V2.00 ～ V2.13 (統合開発環境 PM+)

3.2 内容

near属性の配列の要素、near属性の構造体 または 共用体メンバに対し、farでキャストして連続アクセスすると誤ったコードとなる場合があります。

3.3 発生条件

以下のすべての条件を満たす時に発生する場合があります。

(1) 関数内で、自動変数 または 仮引数を使っている。
(2) 静的にnear配置した配列の要素、near配置した構造体 または 共用体メンバにアクセスしている。
(3) (2)のアクセスの後に、アセンブラコード上で「(2)のアドレス+1」にある配列要素構造体 または 共用体メンバに対し、アドレスを「1byteデータへのfarポインタ」にキャストして間接参照でロードしている(ただし(2)のアクセスの後に関数呼び出しやasm文がある場合は除く)。
(4) (3)のアクセスの後に、「(3)のアドレス+オフセット」にある配列要素構造体 または 共用体メンバをアクセスしている(ただし(3)のアクセスの後に関数呼び出しやasm文がある場合は除く)。
オフセットはアセンブラコード上で-2から+255の間である。
(5) (2)(3)(4)の配列要素、構造体 または 共用体メンバが、同じ識別子である。

発生例

```
------------------------------------------------------------
unsigned char __near const uca1[7] = { 1, 2, 3, 4, 5, 6, 7 };
unsigned char __near uc1, uc2, uc3;

#define data0   (*(unsigned char __far *)&uca1[0])
#define data1   (*(unsigned char __far *)&uca1[1])
#define data2   (*(unsigned char __far *)&uca1[2])
#define data3   (*(unsigned char __far *)&uca1[3])
#define data4   (*(unsigned char __far *)&uca1[4])
#define data5   (*(unsigned char __far *)&uca1[5])
#define data6   (*(unsigned char __far *)&uca1[6])

void func(void)
{
   unsigned char dummy;  /* 発生条件(1) */
   uc1 = data2;  /* 発生条件(2)(5) */
            /* uca1[2]をfarアドレスの間接参照でアクセス */
   uc2 = data3;  /* 発生条件(3)(5) */
            /*  uca1[2+1]をfarアドレスの間接参照でアクセス */
   uc3 = data4;  /* 発生条件(4)(5) */
            /*  uca1[3+1]をfarアドレスの間接参照でアクセス */
}
------------------------------------------------------------
```

上記発生例の場合の出力コード例:

```
------------------------------------------------------------
; line    14 :     uc1 = data2;
movw    de,#mirlw (_uca1+2)
mov     a,[de]
mov     !_uc1,a
; line    15 :     uc2 = data3;
mov     a,[de+1]
mov     !_uc2,a
; line    16 :     uc3 = data4;
mov     a,[de+1]    ; uca1[4]ではなくuca1[3]をアクセス
mov     !_uc3,a
------------------------------------------------------------
```

3.4 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)のアクセスをnearポインタへのキャストに変更する。

回避例1:

```
------------------------------------------------------------
#define data3    (*(unsigned char __near *)&uca1[3])
------------------------------------------------------------
```

(2) 発生条件(3)のアクセスをキャストなしに変更する。

回避例2:

```
------------------------------------------------------------
#define data3    (uca1[3])
------------------------------------------------------------
```

4. strcmp関数、strncmp関数の戻り値が不正となる注意事項

4.1 該当製品 および バージョン

CA78K0R V1.70 ～ V1.71 (統合開発環境 CS+)
CA78K0R V1.20 ～ V1.70 (統合開発環境 CubeSuite+)
CA78K0R V1.00 ～ V1.10 (統合開発環境 CubeSuite)
CC78K0R V1.00 ～ V2.13 (統合開発環境 PM+)

4.2 内容

strcmp関数、strncmp関数を使用して引数を比較する際に、戻り値が不正となる場合があります。

4.3 発生条件

以下のすべての条件を満たす時に発生します。

(1) 以下のいずれかの関数を使用して引数の比較を行っている。
- strcmp(s1, s2)
- strncmp(s1, s2, n)

(2) s1で指される文字列とs2で指される文字列の1バイトずつの順次比較で、最初の異なる文字対が、下記のいずれかを満たす。
(a)s1で指される文字列の文字コードが0x80以上で、かつ、
s2で指される文字列の文字コードとの差が0x80以上である。
(b)s2で指される文字列の文字コードが0x80以上で、かつ、
s1で指される文字列の文字コードとの差が0x80より大きい。

発生例

```
------------------------------------------------------------
#include
int x1, x2, x3;
void func(void)
{
   x1 = strcmp("¥xc0", "¥x3e"); /* 発生条件(1)(2) */
                                /* x1の値が負となる */
   x2 = strcmp("¥xc0", "¥x40"); /* 発生条件(1)(2) */
```

```
                                    /* x2の値が負となる */
    x3 = strcmp("¥x40", "¥xc2"); /* 発生条件(1)(3) */
                                    /* x3の値が正となる */
}
```

4.4 回避策

ありません。

5. strtoul関数の戻り値が不正となる注意事項

5.1 該当製品 および バージョン

CA78K0R V1.70 ～ V1.71 (統合開発環境 CS+)
CA78K0R V1.20 ～ V1.70 (統合開発環境 CubeSuite+)
CA78K0R V1.00 ～ V1.10 (統合開発環境 CubeSuite)
CC78K0R V1.00 ～ V2.13 (統合開発環境 PM+)

5.2 内容

strtoul関数を使用して文字列を整数に変換する際に、戻り値 および errnoが
不正となる場合があります。

5.3 発生条件

以下のすべての条件を満たす時に発生します。

(1) 以下の関数を使用して文字列を整数に変換している。
    - strtoul(nptr, endptr, base)
(2) 変換対象文字列nptrがマイナス符号付きの文字列である。

発生例

```
#include
char *endptr;
unsigned long ans1, ans2;
void func(void)
{
    /* 発生条件(1)(2)ans1の値が4294966062ではなく0となる */
    ans1 = strtoul("-1234", &endptr, 10);
    /* 発生条件(1)(2)ans2の値がULONG_MAXではなく0となり、 */
    /* errnoの値がERANGEではなく0となる*/
    ans2 = strtoul("-4294967300", &endptr, 10);
}
```

5.4 回避策

ありません。

6. 恒久対策

上記5件の注意事項は、次期バージョンで改修する予定です。
