RENESAS ツールニュース

RENESAS TOOL NEWS 2009年09月01日：090901/tn1

SuperH RISC engineファミリ用C/C++ コンパイラパッケージ V.9 ご使用上のお願い

SuperH RISC engine ファミリ用C/C++コンパイラパッケージ V.9の使用上の注意事項 を連絡します。

- ループ内で、ループ帰納変数とループ不変変数からなる添え字式 または乗算式を記述した場合の注意事項 (SHC-0080)

---

1. 該当製品

SuperH RISC engine ファミリ C/C++コンパイラパッケージ
V.9.00 Release 00 ～ V.9.03 Release 00

2. C/C++コンパイラ

2.1 ループ内で、ループ帰納変数とループ不変変数からなる添え字式または乗算式を記述した場合の注意事項 (SHC-0080)

2.1.1 現象

ループ内で、ループ帰納変数とループ不変変数のみを添え字に持つ、
同じ配列の異なる要素の参照を行った場合、またはループ帰納変数とループ不変変数の乗算を行った場合に、配列アドレスまたは乗算結果が正しくない場合があります。(注1)

注1. ループ帰納変数は、ループを繰り返すごとに値が一定に増加または減少する変数です。

2.1.2 発生条件：

2.1.3項 発生条件1、または 2.1.4項 発生条件2 を満たした場合に発生することがあります。

2.1.3 発生条件1

同じ配列の異なる要素の参照で配列アドレスの計算結果が正しくない場合の条件
(1) optimize=1オプションを使用している。

(2) signed char型またはunsigned char型の配列があり、ループ内で、
2つ以上の異なる要素を参照している。
(3) (2)の配列要素を参照する添え字は、1つのループ帰納変数と1つの
ループ不変変数のみで構成されている。
ただし、以下の例のようにコンパイラの最適化により、別の変数が、
ループ帰納変数とループ不変変数に置き換わる場合も含む。

例
```
-----------------
 k=i+x;
 a[k]; // a[i+x]
-----------------
```

(4) (3)のループ帰納変数の型は以下のいずれかである。
char型、 unsigned char型、signed short型、
unsigned short 型、signed int型、unsigned int型、
signed long型 および unsigned long型
(5) (3)のループ不変変数の型は以下のいずれかである。
signed char型、unsigned char型、signed short型 および
unsigned short型
(6) (2)の配列要素の参照における、(3)のループ帰納変数の増分値または
減分値は同じである。

参考情報
不具合に該当するコンパイラの生成コードは下記(a)～(d)をすべて
満たします。上記発生条件に当てはまる場合、実際に不具合が再現
するかどうかは、以下のコンパイラ生成コードの条件をご確認ください。
(a) コンパイラの生成コードで、発生条件(2)の配列要素の参照のうち、
少なくともひとつは他の配列要素のアドレスとオフセットを用いて
その配列要素のアドレスを計算している。
(b) (a)のオフセットは2つの配列要素のループ不変変数をオペランド
に持つSUB命令になっている。
(c) (b)のSUB命令は、発生条件(2)のループの前に出力されている。
(d) (b)のSUB命令の結果は、参考情報(b)の2つのループ不変変数のうち
大きい方の型で表現できない値である。(注2) (注3)

注2. 型の大小関係は以下のように判定されます。
unsigned short > signed short > unsigned char > signed char

注3.「型で表現できない値」は、以下の範囲の値を指します。
signed char : -129以下 または 128以上
unsigned char : 負の値 または 256以上
signed short : -32,769以下 または 32,768以上

unsigned short： 負の値 または 65,536以上

発生例：

```
-----------------------------------------------
signed char a[100000];  // 発生条件(2)
unsigned short x = 1;   // 発生条件(5)
unsigned short y = 5;   // 発生条件(5)

f()
{
   int i;              // 発生条件(4)
   for (i=0;i<10;i++) { // 発生条件(6)
      a[i+y] = a[i+x]; // 発生条件(2)および(3)
   }
}
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------

 _f:
      MOV.L    L13+2,R1      ; _x
      MOV.L    L13+6,R2      ; _y
      MOV.W    @R1,R4        ; x
      MOV.W    @R2,R1        ; y
      MOV.L    L13+10,R6     ; _a
      SUB      R1,R4         ; 参考情報(b)(c)(d)
                             R4=x-y
      EXTU.W   R1,R1
      ADD      R1,R6         ; R6=&a[i+y]
      EXTU.W   R4,R4         ; 2バイトゼロ拡張でSUB
                             命令の結果が変わる
      MOV      #10,R5        ; H'0000000A
L11:
      MOV      R4,R0         ; R0 = (unsigned
                             short)(x-y)
      MOV.B    @(R0,R6),R2   ; 参考情報(a) (a[i+x]に
                             対応、
```

```
                                        &a[i+y]+(x-y)でアク
                                       セス)
        ADD     #-1,R5
        TST     R5,R5
        MOV.B   R2,@R6          ; a[]
        ADD     #1,R6
        BF      L11
        RTS
        NOP
```

2.1.4 発生条件2

ループ帰納変数とループ不変変数の乗算の結果が正しくない場合の条件

(1) optimize=1オプションを使用している。

(2) ループ内にループ帰納変数とループ不変変数の乗算が存在する。

(3) (2)のループ帰納変数の増分値または減分値はループ不変変数である。

(4) (2)のループ帰納変数の型は以下のいずれかである。

char型、 unsigned char型、signed short型、
unsigned short 型、signed int型、unsigned int型
signed long型、またはunsigned long型

(5) (2)のループ不変変数、および(3)のループ不変変数の型は
以下のいずれかである。
signed char型、unsigned char型、signed short型、または
unsigned short型

参考情報

不具合に該当するコンパイラの生成コードは、以下(a)および(b)を満たします。

上記発生条件に当てはまる場合、実際に不具合が再現するかどうかは、
以下のコンパイラ生成コードの条件をご確認ください。

(a) 発生条件(2)のループ不変変数と発生条件(3)のループ不変変数を
オペアランドに持つ乗算命令が、該当ループの前に出力されている。

(b) (a)の乗算の結果は、発生条件(2)および(3)のループ不変変数のうち
大きい方の型で表現できない値である。(2.1.3項の注2および注3参照)

発生例：

Cソース>

```
int S = 0;
```

```
char n = 64;    // 発生条件(5)
char m = 4;     // 発生条件(5)
void main(void)
{
   int i;              // 発生条件(4)
   for (i=0;i<128;i+=n){ // 発生条件(3)
      S += i*m;        // 発生条件(2)
   }
}
// ループは2回まわるので、本来は、S = 0*m + 64*m = 256 と
// なるはずが、S = (char)(0*m) + (char)(64*m) = 0 となってしまう
```

コンパイル結果：

```
_main:
        MOV.L   R14,@-R15
        STS.L   MACL,@-R15
        MOV.L   L14+2,R14       ; _m
        MOV     #0,R6           ; H'00000000
        MOV.L   L14+6,R2        ; _n
        MOV     R6,R5
        MOV.B   @R14,R7         ; m
        MOV.L   L14+10,R14      ; _S
        MOV.B   @R2,R1          ; n
        MULS.W  R7,R1           ; 参考情報(a)(b)
        MOV     #-128,R7        ; H'FFFFFF80
        STS     MACL,R4
        MOV.L   @R14,R2         ; S
        BRA     L11
        EXTU.B  R7,R7
L12:
        EXTS.B  R4,R4           ; n*mを1byte符号拡張
                                するので、
                                  R4=(char)(64*4)=0
```

```
        ADD     R5,R2           ; R2(=S)にR5(=i*m)を
                                  加算。R5は0なので、
                                  R2は元の値(=0)のま
                                  ま。
        ADD     R4,R5           ; R5(=i*m)に
                                  R4(=n*m)を加
                                  算。R4=0なので、
                                  R5は元の値(=0)のま
                                  ま。
        ADD     R1,R6
L11:
        CMP/GE  R7,R6
        BF      L12
        MOV.L   R2,@R14         ; S
        LDS.L   @R15+,MACL
        RTS
        MOV.L   @R15+,R14

-----------------------------------------
```

2.1.5 回避策

以下のいずれかの方法で回避してください。

(1) optimize=1の代わりに、optimize=0またはoptimize=debug_only オプションを使用する。

(2) ループ帰納変数、またはいずれか1つのループ不変変数をvolatile修飾する。

3. 恒久対策

本内容は、SuperH RISC engine ファミリ用 C/C++コンパイラパッケージ V.9.03 Release 01で改修する予定です。