RENESAS ツールニュース

RENESAS TOOL NEWS 2005年06月16日：RSO-SHC-050616D

SuperH RISC engineファミリ 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 C/C++コンパイラパッケージ V.9の使用上の注意事項8件を連絡します。

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.9.00 Release 02

2. **内容**

   2.1 スタックで渡される引数を持つ関数を呼び出す場合の注意事項(SHC-0030)
   スタックで渡される引数を持つ関数を呼び出す際に、その引数の値がスタック 領域内のアドレス値だったとき、誤ったアドレス値が渡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した場合に発生することがあります。
   1. optimize=1オプションを使用している。
   2. スタックで渡される引数を持つ関数を呼び出している。
   3. 2の関数呼び出しの引数にローカル変数のアドレス値のようなスタック 領域内のアドレス値がある。
   4. 3で渡すアドレス値が関数入り口でレジスタ退避後のスタックポインタの値に 128を加えた値よりも小さい。
   5. 3で渡すアドレス値がレジスタに割り付かず、関数呼び出しの直前で アドレス値を計算するコードを生成している。

   例：

```
------------------------------------------------
int sub(int*, int, int, int, int);
int sub2(int, int, int*, int, int);
int data1[10],data2[10],data3[10],data4[10];
void func(int c) {
    int i;
```

```
    int a[1]={0},b[10];
    for (i = 0 ; i < 10 ; i++) {
        sub(b,2,3,4,5);
        sub(b,2,3,4,5);
        if (c) {
            sub2(1,2,b,4,5);
        }
        data1[i] = data2[i] + data3[i] + data4[i];
        data2[i] = data3[i] + data4[i];
    }
}
```

コンパイル結果：

```
_func:
          :
     MOV     #5,R2
     MOV.L   R2,@-R15
     MOV     R15,R4
     MOV     #4,R7
     MOV     #3,R6
     MOV     #2,R5
     JSR     @R8
     ADD     #4,R4   ; bのアドレスはR15+8だがR15+4としている
          :
```

回避策：以下のいずれかの方法で回避してください。

1. optimize=0オプションを使用する。
2. 発生条件3のアドレス値を呼び出し側の関数の先頭でvolatile修飾した変数に代入して、この変数を引数に使用する。

2.2 double型およびfloat型の両方のデータを持つ関数についての注意事項(SHC-0031)

double型およびfloat型の両方のデータを持つ関数において、double型データ の下位32bit、またはfloat型データの値が別の値に書き換えられる場合が あります。

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. cpu=sh4、cpu=sh4aまたはcpu=sh2afpuオプションを使用している。
2. fpu=singleまたはfpu=doubleオプションを使用していない。
3. 関数内でfloat型の変数または定数を定義または使用している。
4. 3の関数が以下4a～4cのいずれかの条件を満たす。
   4a. double型の引数を持つ。

4b. double型の引数、またはリターン値を持つ関数を呼び出している。
4c. double型からunsigned long long型またはlong long型への型変換、 あるいはその逆を行っている。

5. 3の変数にFR1,FR3,FR5,FR7,FR9,FR11のいずれかのレジスタが割り付いている。

例：

```
----------------------------------------
float f1,f2,f3,f4;
double d4;
void func(double d1, double d2, double d3) {
   f1 += f1;
   d4 = d3;
   f4 = f1 + f3;
   f4 += f4;
}
----------------------------------------
```

コンパイル結果：

```
----------------------------------------
_func:                  ; d3はDR8に割り付いている
     MOV.L   L11+2,R1       ; _f1
     MOV.L   L11+6,R4       ; H'00000008+_d4
     FMOV.S  @R1,FR9        ; FR9(d3の下位32bit)にf1を代入
     MOV.L   L11+10,R5      ; _f3
     FADD    FR9,FR9
     MOV.L   L11+14,R6      ; _f4
     FMOV.S  FR9,@-R4       ; d3の値をd4へ代入
     FMOV.S  FR8,@-R4
     FMOV.S  @R5,FR8
     FMOV.S  FR9,@R1
     FADD    FR8,FR9
     FMOV.S  FR9,FR8
     FADD    FR9,FR8
     RTS
     FMOV.S  FR8,@R6
----------------------------------------
```

回避策：以下のいずれかの方法で回避してください。

1. fpu=singleまたはfpu=doubleオプションを使用する。
2. 発生条件4aが該当する場合は、その引数をvolatile修飾する。
3. 発生条件4bが該当する場合は、引数、またはリターン値をvolatile修飾した変数に代入してから使用する。
4. 発生条件4cが該当する場合は、double型変数または定数をvolatile修飾した変数に代入してから使用する。

2.3 strcpy()関数呼び出しまたはpacked構造体メンバへの代入式を含む関数の戻り値に関する注意事項(SHC-0032)

strcpy()関数呼び出しまたはpacked構造体メンバへの代入式を含む関数に おいて、この関数のリターン値が間違った値になる場合があります。

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. cpu=sh2aまたはcpu=sh2afpuオプションを使用している。
2. optimize=1オプションを使用している。
3. リターン値が整数型またはポインタ型である(cpu=sh2aを使用した場合は float型も含む)関数が存在する。
4. 3の関数のreturn文の直前で、strcpy()関数呼び出しもしくはpacked構造体 メンバへの代入を行っている。
5. 3の関数のリターン値は4の処理に依存しない値である。

例：

```
-----------------------------------------
#include <string.h>
char *a,*b;
int func() {
   strcpy(a,b);
   return (0);
}
-----------------------------------------------
 コンパイル結果：
-----------------------------------------------
_func:
     MOV.L   L11+2,R6       ; _a
     MOV.L   L11+6,R7       ; _b
     MOV.L   @R6,R0
     MOV     #0,R2          ; リターン値をR2に設定
     MOV.L   L11+10,R4      ; __slow_strcpy
     JMP     @R4            ; JSRをJMPに置換
     MOV.L   @R7,R1
                        ; PRの退避・回復とRTV/N R2を削除
-----------------------------------------
```

回避策：以下のいずれかの方法で回避してください。

1. 発生条件に該当する関数内でダミーのvolatile変数を定義する(定義のみでよい)。
2. return文の直前にnop()組み込み関数を追加する。
3. optimize=0オプションを使用する。

2.4 ループ内でポインタ変数を含む積和演算を行う場合の注意事項(SHC-0033)

for、while、およびdo-while文によるループ内でポインタ変数を含む 積和演算を行った場合、誤ってMAC.W命令を生成する場合があります。

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. optimize=1オプションを使用している。
2. cpu=sh1、cpu=sh2a、またはcpu=sh2afpuオプションを使用していない。
3. for、while、およびdo-while文によるループが存在する。
4. 3のループ内に積和演算が存在する(乗算と加算が別式の場合も含む)。
5. 4の乗数は共にshort型変数で、加算結果はunsigned long, long, unsigned int, またはint型の変数へ代入されている。
6. 4の乗数はポインタ変数または配列変数を代入したものである。
7. 3のループ内に6のポインタ変数または配列変数へ値を代入する式が存在する。
8. 7の代入は6の代入の前にある。

例：

```
----------------------------------------
#include <stdio.h>
#define MAX (10)
short A[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
short B[] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
short mac2() {
   int i,sum=0;
   short *p1=A;
   short *p2=p1;
   short *q=&(B[MAX-1]);
   short *r=p1;
   short t, s;
   r++;
   sum = *q-- * *p1++;
   for(i=0; i < (MAX-1); i++){
      *p2++ = *p1;
      t = *q--;
      *r++ = 2;       // (A) rとp1は同じアドレスA[]を指す
      s = *p1++;
      sum += (t * s);  // sum += (*q++ * *p1++);と置き換え、
                  // (A)で変更後の値を使用する
   }
   return (sum);
}
----------------------------------------
```

回避策：以下のいずれかの方法で回避してください。

1. optimize=0オプションを使用する。
2. 発生条件4の乗算式の左辺式または右辺式を乗算前にvolatile修飾した変数に代入する(使用する必要はない)。

2.5 条件式の比較結果を代入した変数から1を減算している場合の注意事項(SHC-0034)

条件式の比較結果を代入した変数から1を引く減算式があるとき、代入と 減算式の間に関数呼び出しがあると、減算結果が正しくない場合があります。

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. optimize=1オプションを指定している。
2. 条件式の結果を変数に代入している。
3. 2の変数から1を引く減算式が存在する。
4. 2の代入と3の減算式の間に、戻り値を持たない、または戻り値がR0でない関数呼び出しがある。

例：

```
----------------------------------------
int X, Y, Z;
int A, B, C, D, E, F;
int foo() {
   return(0);
}
void bar(int a, int b, int c, int d, int e) {}
test() {
   int t, m, n, r, s, u, v, w;
   t = E;
   s = A;
   u = B;
   v = C;
   w = D;
   F = t;
   r = E;
   t = (X == 0);
   n = (Y == 0);
   m = (Z == 0);
   (void)foo();
   t = t - 1;
   n = n - 1;
   m = m - 1;
   t = -t;
   n = -n;
   m = -m;
```

```
    Y = t;
    Z = n;
    X = m;
    bar(s, u, v, w, r);
    B = s;
    C = u;
    D = v;
    E = w;
    A = r;
}
-------------------------------------------
 コンパイル結果：
-------------------------------------------
_test:
          :
    SUBC    R0,R0          ; (X == 0)の結果をR0に格納
    MOV.L   L21+32,R1      ; _Z
    MOV.L   @R1,R5
    TST     R5,R5
    MOVT    R6
    MOV.L   R6,@(4,R15)
    BSR     _foo           ; 関数呼び出し先でR0の値を破壊
    NOP
    MOV.L   @R15,R2
    ADD     #-1,R2
    ADD     #1,R0          ; 破壊後のR0の値を参照
    MOV.L   @(4,R15),R6
    ADD     #-1,R6
    NEG     R2,R2
    MOV     R0,R14
          :
-----------------------------------------
```

回避策：以下のいずれかの方法で回避してください。

1. optimize=0オプションを使用する。
2. 条件式の結果を代入する変数をvolatile修飾する。
3. 発生条件4の関数を戻り値を返すようにし、その戻り値をダミーの変数に代入する。

2.6 long long型である2のべき乗を除数とする剰余演算の結果に関する注意事項(SHC-0035)

long long型またはunsigned long long型の2のべき乗を除数とする剰余演算の結果が正しくない場合があります。

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. 除数または被除数がlong long型またはunsigned long long型の剰余演算が存在する。
2. 1の被除数は、以下の2a、2bのいずれかに該当する式である。
   2a. 符号無し整数型の変数または符号無し整数型のビットフィールドメンバである。
   2b. 符号付き整数型の変数または符号付き整数型のビットフィールドメンバであり、かつ1の剰余演算は等価演算子の第1オペランドまたは第2オペランドであり、かつその等価演算のもう一方のオペランドは定数0である。
3. 1の除数は、0x100000000以上0x8000000000000000以下の2のべき乗の定数である。

例：

```
----------------------------------------
unsigned long long X;
long long Y;
void func1(void) {
   X = X % 0x0000000800000000ull;
      // 誤ってX & 0xFFFFFFFFFFFFFFFFullに置換
}
void func2(void) {
   if ((Y % 0x0000000800000000ll) == 0) {
      // 誤ってY & 0xFFFFFFFFFFFFFFFFll に置換
      func1();
   }
}
----------------------------------------
```

回避策： 発生条件に該当する除数をvolatile修飾した変数に代入して、この変数を剰余演算の除数とする。

例：

```
----------------------------------------
void func1() {
   volatile unsigned long long c=0x000000800000000ull;
   X = X % c;
}
----------------------------------------
```

2.7 乗算の演算結果をその被乗数または乗数と同じ整数定数で除算している場合の注意事項(SHC-0036)

式と整数定数を乗算した結果を同じ整数定数で除算したときに、演算結果が 正しくない場合が

あります。

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. 符号無し整数型の式と0以外の整数定数の乗算式が存在する。
2. 1の乗算結果を1の整数定数で除算している。
3. 1の乗算結果の値がその乗算式の型の最大値を超える。

例：

```
----------------------------------------
unsigned int a=65536;
unsigned int b;
void func() {
   b=(65536*a)/65536;  // b=0 ((65536*65536)/65536→0/65536=0) が
                       // 正しい結果だが、b=a (=65536)に
                       // 置き換えられる
}
----------------------------------------
```

回避策： 発生条件に該当する乗算式の乗算結果をvolatile修飾した
変数に 代入して、この変数を除算の被除数とする。

例：

```
----------------------------------------
void func() {
   volatile unsigned int t=65536*a;
   b=t/65536;
}
----------------------------------------
```

2.8 同一の配列への代入文がif文の前およびif文内に存在する場合の注意事項(SHC-0037)

if文の前とこのif文と対応するthen節内に同じ配列への代入文が存在する とき、代入が正しく行われない場合があります。

発生条件：以下の条件をすべて満たした場合に発生することがあります。

1. optimize=1オプションを使用している。
2. for, while, do-while文によるループが存在する。
3. 2のループ内にelse節を持たないif文が存在する。
4. 3のif文の前とthen節内に同じ配列への代入文が存在する。 以下、if文の前の代入文の右辺式をexp1、if文のthen節内の代入文の 右辺式をexp2とする。
5. exp1とexp2は異なるサイズの型を持つ。
6. 4のexp1とexp2が以下の6a、6bのいずれかに該当する。
   6a. exp1がexp2の型の範囲外の値を取ることがある。

6b. exp2がexp1の型の範囲外の値を取ることがある。

例：

```
-----------------------------------------
int A[10];
int x, y, z;
char sc0, sc1, sc2;
test() {
   int i;
   for(i = 0; i < 10; i++){
      A[i] = x + y;    // if(z)がFALSEになるときに配列A[i]に
      if(z){           // x+yの値が正しく代入されない
         A[i] = sc0;
      }
      x++;
      y++;
      sc0++;
   }
}
-----------------------------------------
```

回避策：以下のいずれかの方法で回避してください。

1. optimize=0オプションを使用する。
2. 発生条件3のif文をelse節を持つif文に書き換え、そのelse節にnop()組み込み関数を挿入する。
3. 発生条件3のif文のthen節内にnop()組み込み関数を挿入する(挿入箇所はthen節内であればどこでも良い)。
4. 発生条件4のexp1と発生条件3のif文の間にnop()組み込み関数を挿入する。

3. 恒久対策

本内容は、SuperH RISC engine ファミリ C/C++コンパイラパッケージ V.9.00 Release 03 で改修する予定です。