RENESAS ツールニュース

RENESAS TOOL NEWS 2006年03月01日：RSO-SHC-060301D

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 12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 03

2. **内容**

   2.1 **浮動小数点定数を複数使用した場合の注意事項(SHC-0052)**

   同一ブロック内で浮動小数点定数を複数使用した場合に、正しくない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) 以下(a)(b)のいずれかの条件を満たす。

   (a) cpu=sh2e オプションを使用している。

   (b) cpu=sh2afpu, cpu=sh4, またはcpu=sh4a オプションの何れかひとつを使用し、かつ fpu=single オプションを使用している。

   (3) 同一ブロック内で浮動小数点定数を以下の(a)から(e)のいずれかの順番で使用している。 各浮動小数点定数を使用している行の間に別の行があっても該当します。

	1番目	2番目

(a)	0.0f	1.0f
(b)	0.0f	-1.0f
(c)	0.0f	2.0f
(d)	1.0f	0.0f
(e)	1.0f	-0.0f

例：

```
-------------------------------------------
float f1,f2;
void func() {
       :
   f1 = 0.0f;  // 発生条件(3)(a)
   f2 = 1.0f;  // 発生条件(3)(a)
      :
}
-------------------------------------------
```

コンパイル結果：

```
-------------------------------------------
_func:
            :
        MOV.L      L11+2,R1   ; _d1
        MOV.L      L11+6,R4   ; _d2
        FLDI0      FR8
        FMOV.S     FR8,@R1
        ADD        #0,FR8     ; 正しくない命令を生成
        FMOV.S     FR8,@R4
            :
-------------------------------------------
```

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)の浮動小数点定数のどちらか一方の値を初期値とする外部変数を定義して、定数をその外部変数に置き換える。

(2) optimize=0 オプションを使用する。

2.2 積和演算を行った場合の注意事項(SHC-0053)

繰り返し文のループ本体で積和演算を行った場合に、積和演算の結果が正しくない(現象1)か、 または関数の入口および出口でMACHレジスタの退避および回復を生成しない(現象2)場合

があります。

発生条件： 現象1は(1)から(7)の条件をすべて満たした場合に、現象2は(1)から(6)、(8)、 および(9)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 オプション以外を使用している。

(3) 繰り返し文が存在する。

(4) (3)の繰り返し文のループ本体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)の積和演算の乗数は共にsigned short型、signed int型またはsigned long型でかつ同じ型である。

(6) (4)の乗数はポインタ変数または配列変数である。

(7) (3)の繰り返し文のループ本体に(4)の積和演算以外にMACHレジスタ、 MACLレジスタの少なくとも一方を更新する命令が存在する。(現象1が発生する場合のみ該当)

(8) macsave=0 オプションを使用していない。(現象2が発生する場合のみ該当)

(9) (3)の繰り返し文がある関数に以下(a)および(b)のいずれの式も存在しない。
(現象2が発生する場合のみ該当)

(a) 組み込み関数 macw(), macwl(), macl(), macll(), dmulu_h(), dmulu_l(), dmuls_h(), およびdmuls_l()

(b) 乗数がsigned char型でもunsigned char型でもない整数型で、 かつ少なくとも一方の乗数をsigned long long型またはunsigned long long型に型変換している乗算式

現象1の例：

```
-----------------------------------------------
long d[10],e[10];
long long sum;
int func1(short *p, short *q) {
  int i,ret=0;
  for(i=0;i<10;i++) {              // 発生条件(3)
```

```
      ret += *p++ * *q++;            // 発生条件(4) ～ (6)
      sum += (long long)d[i] * e[i];  // 発生条件(7)
   }
   return ret;
}
```

コンパイル結果

```
_func1:
            :
        MOV        #0,R2
        LDS        R2,MACL
            :
L11:
        MAC.W      @R10+,@R11+ ; 積和演算の結果を
MACLに保持
        MOV.L      @(4,R9),R1
        MOV.L      @R9,R7
        MOV.L      R1,@-R15
        MOV.L      R7,@-R15
        ADD        #-8,R15
        MOV.L      @R13+,R1
        MOV.L      @R12+,R4
        DMULS.L    R1,R4       ; MACLレジスタを更新
        STS        MACH,R2
        STS        MACL,R5
        MOV.L      R2,@R15
        MOV.L      R5,@(4,R15)
        JSR        @R8
        MOV.L      R9,@-R15
        DT         R14
        BF/S       L11
        ADD        #20,R15
        STS        MACL,R0     ; 正しくない積和演算の演算結
果を取得
            :
```

現象2の例：

```
int func2(int *p, int *q) {
   int i,ret=0;
   for(i=0;i<10;i++) {         // 発生条件(3)
      ret += *p++ * *q++;     // 発生条件(4) ～ (6)
```

```
    }
    return ret;
}
```

コンパイル結果

```
_func1:
        STS.L       MACL,@-R15
                            ; MACHレジスタの退避命令を生成し
ない
        MOV         #0,R6
        LDS         R6,MACL
        MOV         #10,R2
L11:
        DT          R2
        BF/S        L11
        MAC.L       @R4+,@R5+   ; MACHレジスタを更新
        STS         MACL,R0
                            ; MACHレジスタの回復命令を生成し
ない
        RTS
        LDS.L       @R15+,MACL
```

回避策：以下のいずれかの方法で回避してください。

(1) 積和演算で使用する変数のいずれか1つをvolatile修飾した変数に代入して、その変数を積和演算に使用する。

(2) optimize=0 オプションを使用する。

(3) macsave=0 オプションを使用する。(現象2に該当する場合のみ有効)

2.3 do-while文を使用した場合の注意事項(SHC-0054)

繰り返し回数が1回のdo-while文が、2回以上繰り返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) do-while文が存在する。

(3) (2)のループ本体にsigned int, unsigned int, signed long, またはunsigned long型のループ制御変数が存在する。

(4) (2)のdo-while文のループ制御変数の初期値、増分値、および制御式が以下の(a)または(b)のいずれかである。

(a) 初期値が0、増分値が1でかつ制御式が
(ループ制御変数) < (定数以外の式) または
(定数以外の式) > (ループ制御変数) の形の
比較式である。

(b) 初期値が1、増分値が1でかつ制御式が
(ループ制御変数) <= (定数以外の式) または
(定数以外の式) >= (ループ制御変数) の形の
比較式である。

(5) (4)の(定数以外の式)の計算結果は0以下の整数値である。

例：

```
-----------------------------------------------
unsigned long N = 0;          // 発生条件(5)
void func(void) {
   unsigned long count = 0;  // 発生条件(3) and
(4)(a)
   do {                    // 発生条件(2)
      count++;
   } while (count < N);      // 発生条件(4)(a)
}
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------
_func:
        MOV.L      L13+2,R2   ; _N
        MOV.L      @R2,R2     ; R2に0を設定
L11:
        DT         R2         ; 1回目のループでR2は
0xFFFFFFFFとなり
                              ; Tビットは0となる
        BF         L11        ; 1回目のループでTビット
は0になるため、
                              ; L11に分岐する
        RTS
        NOP
```

回避策：以下のいずれかの方法で回避してください。

(1) 初期値、増分値、または制御式のいずれかを発生条件(4)を満たさないように変更する。

(2) do-while文をfor文またはwhile文に変更する。

(3) 発生条件(3)のループ制御変数の型を、signed char, unsigned char,signed short, またはunsigned short に変更する。

(4) 発生条件(3)のループ制御変数をvolatile修飾する。

(5) optimize=0 オプションを使用する。

2.4 1つの関数内に同じ式が複数存在する場合の注意事項(SHC-0055)

1つの関数内に同じ式が複数存在し、かつその式の演算で実行時ルーチンを使用する場合、 実行時ルーチンによる演算結果を正しくないスタック領域に退避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) 関数内に同じ式が2つ以上存在する。

(3) (2)の式は以下の(a)から(d)の変数もしくは定数を少なくとも1つ含んでいる。

  (a) long long型変数またはunsigned long long 型変数

  (b) -2147483648から2147483647の範囲外の整数定数

  (c) double型変数またはdouble型定数
  ただし以下のいずれかのオプションを使用している場合は対象外
  cpu=sh2afpu, cpu=sh4, cpu=sh4a, または double=float

  (d) long double型変数またはlong double型定数
  ただし以下のいずれかのオプションを使用している場合は対象外
  cpu=sh2afpu, cpu=sh4, またはcpu=sh4a

例：

```
-------------------------------------------
extern void g();
unsigned long b[4];
void func() {
   static long c[4][4][4];
   long d[4];
   int i,j,k;
   for(i=0; i<4; i++) {
      for(j=0; j<4; j++){
         for(k=0; k<4; k++){
            c[i][j][k] = 0;
         }
      }
   }
   for(i=0; i<4; i++) {
      d[i]=0;
   }
   for(i=0; i<4; i++) {
      b[i] = 2147483648;          // 発生条件(2) and
(3)(b)
   }
   for(i=0; i<4; i++) {
      if (b[i] == 2147483648u) {    // 発生条件(2) and
(3)(b)
         g();
      }
   }
}
-------------------------------------------
```

コンパイル結果：

```
-------------------------------------------
_func:
        MOV.L      R8,@-R15
        MOV.L      R9,@-R15
        MOV.L      R10,@-R15
        MOV.L      R11,@-R15
        MOV.L      R12,@-R15
        MOV.L      R13,@-R15
        MOV.L      R14,@-R15
        STS.L      PR,@-R15
        ADD        #-12,R15
```

```
        MOV       #-128,R1   ; H'FFFFFF80
        SHLL8     R1
        SHLL16    R1
        MOV.L     R1,@-R15
        MOV       #0,R4      ; H'00000000
        MOV.L     L24+2,R5   ; __conv64u
        JSR       @R5
        MOV.L     R4,@-R15
        MOV.L     R0,@R15    ; 正しくはMOV.L
R0,@(8,R15)
        ADD       #8,R15
            :
-----------------------------------------------
```

回避策：optimize=0 オプションを使用してください。

2.5 仮引数を除算および剰余算で使用した場合の注意事項(SHC-0056)

仮引数を除算および剰余算の除数または被除数に使用した場合に、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) 除算または剰余算が存在する。

(2) (1)の除算および剰余算の除数または被除数の少なくとも一方は仮引数である。

(3) (2)の仮引数の型はsigned char, unsigned char, signed short, またはunsigned shortである。

(4) (1)の除算および剰余算の除数と被除数の型が異なる。

例：

```
-----------------------------------------------
short func(unsigned short y) {  // 発生条件(3)
   short x=-200;
   x /= y;                  // 発生条件(1), (2), and
(4)
   return x;
}
-----------------------------------------------
```

コンパイル結果：

```
-------------------------------------------
_func:
        MOV.L       L11+4,R2   ; __divwu
        MOV.W       L11,R1     ; H'FF38
        JMP         @R2        ; 符号付き4バイト除算が
正しいが、
                               ; 符号なし2バイト除算が行わ
れる
        EXTU.W      R4,R0
-------------------------------------------
```

回避策：以下のいずれかの方法で回避してください。

(1) 仮引数を除算で使用するときに、仮引数を宣言したときの型に明示的に型変換する(例では、x /= (unsigned short)y;)。

(2) 仮引数を別の変数に代入し、この変数を除算に使用する。

2.6 繰り返し文のループ本体で減算式を使用した場合の注意事項 (SHC-0057)

繰り返し文のループ本体に減算式があり、かつその減算式の第2項に signed char, unsigned char, signed short, または unsigned short型の ループ帰納変数を使用した場合に、減算式の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) optimize=1 オプションを使用している。

(2) 繰り返し文が存在する。

(3) (2)の繰り返し文にsigned char, unsigned char, signed short, またはunsigned short型のループ帰納変数が存在する。

(4) (2)の繰り返し文のループ本体に減算式が存在する。

(5) (4)の減算式の第2項は(3)のループ帰納変数である。

例：

```
-------------------------------------------
int A[10];
```

```
int X = 10;
void func()
{
   unsigned char i;          // 発生条件(3)
   for (i = 0; i < X; i++) {  // 発生条件(2)
      A[2 - i] = 1;          // 発生条件(4), and (5)
   }
}
```

コンパイル結果：

```
_func:
        MOV.L      L14+2,R2   ; _X
        MOV        #0,R5      ; H'00000000
        MOV.L      @R2,R1
        BRA        L11
        MOV        #1,R7      ; H'00000001
L12:
        NEG        R5,R2
        EXTU.B     R2,R0      ; NEG命令の結果を余計にゼロ
拡張
        MOV.L      L14+6,R4   ; _A
        ADD        #2,R0
        SHLL2      R0
        MOV.L      R7,@(R0,R4)
        ADD        #1,R5
L11:
        EXTU.B     R5,R2
        CMP/GE     R1,R2
        BF         L12
```

回避策：以下のいずれかの方法で回避してください。

(1) ループ帰納変数の型をsigned int, unsigned int, signed long, またはunsigned long型に変更する。

(2) 減算式の2つの項を入れ替えて加算式に変更する(例では、A[-i+2])。

(3) ループ帰納変数をvolatile修飾する。

(4) optimize=0 オプションを使用する。

2.7 境界調整数1の構造体または共用体でポインタ型のメンバを宣言した場合の注意事項(SHC-0058)

境界調整数1の構造体または共用体でポインタ型のメンバを宣言し、 そのメンバをstrcpy()関数の引数に使用している場合にstrcpy()の実行結果が正しくない場合があります。 また、メンバを整数型に型変換した後に 演算で使用している場合に、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) unaligned=runtime オプションを使用している。
もしくはsize オプションを使用し、かつunaligned オプションを使用していない。

(2) 構造体または共用体が存在する。

(3) pack=1 オプションを使用している。
もしくは(2)の構造体または共用体は#pragma pack 1を付加して宣言している。

(4) (2)の構造体または共用体はポインタ型のメンバを持つ。

(5) (4)のメンバは以下の(a)または(b)のいずれかの条件を満たす。

(a) strcpy()の引数に使用している

(b) 整数型に型変換した後に乗算、除算、剰余算、またはシフト演算のいずれかの演算で使用している

例：

```
-----------------------------------------------
#include <string.h>
#pragma pack 1                  // 発生条件(3)
struct ST {
   char *string;                // 発生条件(4)
} st;                           // 発生条件(2)
void func (const char *ptr) {
   strcpy(st.string, ptr);      // 発生条件(5)(a)
}
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------
_func:
```

```
        STS.L      PR,@-R15
        MOV.L      L11,R1     ; _st
        MOV.L      L11+4,R3   ; __pack1_ld32
        MOV.L      L11+8,R2   ; __slow_strcpy
        JSR        @R3
        NOP                   ; MOV R4,R1が生成されないため、
                              ; strcpy()の第2引数としてstのアドレスが
                              ; 渡される
        JMP        @R2
        LDS.L      @R15+,PR
-----------------------------------------
```

回避策：以下のいずれかの方法で回避してください。

(1) 該当する構造体または共用体を#pragma pack 1を付加しないで宣言して、かつpack=1 オプションを使用しない。

(2) unaligned=inline オプションを使用する。

2.8 **オーバフロー演算を行った場合の注意事項(SHC-0059)**

演算結果の値が演算結果を代入する変数の型で表現可能な範囲を超える場合、その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) optimize=1 オプションを使用している。

(2) 以下(a)(b)のいずれかの条件を満たす。

(a) (a-1)から(a-4)の条件をすべて満たす。

(a-1) 加算、減算、乗算、左シフト、または単項マイナスのいずれかの演算が存在する。

(a-2) (a-1)の演算結果をsigned char, unsigned char, signed short, またはunsigned short型の変数に代入している。

(a-3) (a-1)の演算結果が(a-2)の型で表現可能な範囲を超える。

(a-4) (a-2)の変数をvolatile修飾していない。

(b) (b-1)から(b-4)の条件をすべて満たす。

(b-1) 右辺が除算式である代入文が存在する。

(b-2) (b-1)の除算式の除数、被除数、および代入先の型はすべて同じ型で、signed char型またはsigned short型である。

(b-3) (b-1)の除算式の除数および被除数とも明示的にunsigned longまたはunsigned intに型変換している。

(b-4) (b-1)の除算式の除数および被除数の型変換前の値が負である。

例：

```
-----------------------------------------------
// Y = -128 が正しいが、Y = 128となる
char X = -128;
int Y, Z = 0;
void func() {
   char t;
   t = 0 - X;  // 発生条件 (2)(a)
   Y = t + Z;
}
-----------------------------------------------
// Z = 0 が正しいが、Z = -1となる
signed char X=2,Y=-2,Z;
void func() {
   Z = (unsigned int)X / (unsigned int)Y;  // 発生条件
(2)(b)
}
-----------------------------------------------
```

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(2)の(a)に該当する場合、発生条件(a-2)の代入先変数をvolatile修飾する。

(3) 発生条件(2)の(b)に該当する場合、 除数または被除数のどちらか一方をunsigned long型またはunsigned int型の変数に代入した後、 その変数を除算式で使用

する。

(4) 発生条件(2)の(b)に該当する場合、除数および被除数ともunsigned shortに型変換する。

2.9 特殊ループを記述した場合の注意事項(SHC-0060)

ループ制御変数がオーバフローするなどの特殊ループの場合、繰り返し回数やループ制御変数の値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 オプションを使用している。

(2) 繰り返し文が存在する。

(3) (2)の繰り返し文は以下の(a)から(g)のいずれかの条件を満たす。

(a) (2)の繰り返し文のループ制御変数が繰り返し中にオーバフローする。

(b) (2)の繰り返し文のループ制御変数の増分値がsigned int型の最小値 (0x80000000)である。

(c) (c-1)から(c-3)の条件をすべて満たす。

(c-1) (2)の繰り返し文のループ制御式は比較式で、かつその式内でループ制御変数を型変換している。

(c-2) (c-1)の比較式の比較演算は!=である。

(c-3) (c-1)の比較式の比較演算(!=)を大小比較演算に置き換えた 場合に、最初の繰り返し判定の真偽が、 ループ制御変数の型変換の有無により逆になる。

(d) (d-1)から(d-3)の条件をすべて満たす。

(d-1) (2)の繰り返し文のループ制御式は比較式で、かつその式内でループ制御変数を型変換している。

(d-2) (d-1)の比較式の比較演算は<, <=,

>=, または > である。

(d-3) (d-1)のループ制御式の最初の繰り返し判定の真偽が型変換の有無で異なる。

(e) (2)の繰り返し文のループ制御変数初期値をI、増分値をS、ループ上限値をFとした場合、 (F-1)-Iまたは((F-1)-I)+Sの計算結果がループ制御変数の型で表現可能な範囲を超える。

(f) (f-1)(f-2)の条件をすべて満たす。

(f-1) (2)の繰り返し文のループ制御変数初期値をI、増分値をS、ループ上限値をFとした場合、 F-1またはF-Sの計算結果が ループ制御変数の型で表現可能な範囲を超える。

(f-2) (2)の繰り返し文のループ本体にif文が存在し、かつその 制御式は、ループ制御変数と定数の比較式である。

(g) (g-1)から(g-3)の条件をすべて満たす。

(g-1) (2)の繰り返し文のループ本体にループ帰納変数を含む乗算式 または左シフト式が存在する。

(g-2) (g-1)の乗算式または左シフト式の演算結果がその演算の型で 表現可能な範囲を超える場合がある。

(g-3) (g-1)のループ帰納変数が(2)の繰り返し文以降で参照される。

(4) (3)のループ制御変数またはループ帰納変数をvolatile修飾していない。

例：

```
-----------------------------------------------
// 1回だけ回るはずが、1回も回らないで抜けてしまう
int a;
void func() {
```

```
    short i;
    a = 0;
    for (i = -129; (char)i != -128; i++) { // 発生条件
(3)(c)
        a++;
    }
}
-----------------------------------------
// 1回も回らないはずが、1回以上回ってしまう
int a;
void func() {
    unsigned int i;
    for (i = 1; i < 0; i++) { // 発生条件(3)(e)
                              // F=0, S=1であり、F-1、F-Sが
                              // unsigned int型の範囲を超える
        if (i < 20) {
            a++;
        }
    }
}
-----------------------------------------
```

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(a)から(f)に該当する場合、繰り返し文のループ制御変数をvolatile修飾する。

(3) 発生条件(3)の(g)に該当する場合、該当する繰り返し文のループ帰納変数をvolatile修飾する。

2.10 opt_range オプションを使用した場合の注意事項(SHC-0061)

opt_range=noblock またはopt_range=noloop オプションを使用した場合、 外部変数に対する最適化がそのオプションの制限を越える範囲で実施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) opt_range=noblock またはopt_range=noloop オプションを使用している。

(3)　繰り返し文が存在する。

(4)　(3)の繰り返し文のループ本体またはループ制御式に外部変数が存在する。

例：

```
-------------------------------------------
// -opt_range=noblock指定
int a[100];
int X,Y,Z;
void func() {
   int i;
   for (i=0; i<100; i++) {  // 発生条件(3)
      a[i] = X+Y;          // 発生条件(4)
      if (X) {
         a[i] = Y+Z;
      }
   }
}
-------------------------------------------
```

最適化後のソースイメージ：

```
-------------------------------------------
int a[100];
int X,Y,Z;
void func() {
   int i;
   for (i=0; i<100; i++) {
      temp = X+Y;
      if (X) {
         temp = Y+Z;
      }
      a[i] = temp;    // a[i]への代入がif文の真偽に関わらず
1回になる
   }
}
-------------------------------------------
```

回避策：optimize=0 オプションを使用してください。

2.11　volatile_loop オプションを使用した場合の注意事項(SHC-0062)

volatile_loop オプションを使用し、かつinfinite_loop=1 オ

プションを使用した場合、 ループ制御式中の外部変数の参照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) volatile_loop オプションを使用している。

(3) infinite_loop=1 オプションを使用している。

(4) 繰り返し文が存在する。

(5) (4)の繰り返し文のループ制御式にループ本体で値が更新される変数が存在しない。

(6) (4)の繰り返し文のループ制御式に外部変数が存在する。

(7) (4)の繰り返し文の前に(6)の外部変数への代入文が存在する。

例：

```
------------------------------------------------
signed int n;
signed int a[100];
void func() {
   n = 0;             // 発生条件(7)
   while (n == 1) {    // 発生条件(4)～(6)
      a[n] = 0;
   }
}
------------------------------------------------
```

コンパイル結果：

```
------------------------------------------------
_func:
        MOV.L       L11,R6     ; _n
        MOV         #0,R2      ; H'00000000
                               ; nの参照およびwhile文が削除される
        RTS
        MOV.L       R2,@R6
------------------------------------------------
```

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(4)の繰り返し文と発生条件(7)の代入文の間に、nop()組み込み関数を挿入する。

(2) infinite_loop=0 オプションを使用する。

(3) optimize=0 オプションを使用する。

2.12 infinite_loop=0 オプションを使用した場合の注意事項 (SHC-0063)

infinite_loop=0 オプションを使用し、かつswitch文およびgoto文により構成 された無限ループが存在する場合、 その無限ループの直前の外部変数アクセス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) infinite_loop=0 オプションを使用している。

(3) switch文とgoto文で構成された無限ループが存在する。

(4) (3)のswitch文の前で外部変数アクセスを行なっている。

(5) (4)の外部変数をvolatile修飾していない。

例：

```
-----------------------------------------------
int a;
void func() {
   a=1;              // 発生条件(4)
Label:
   switch (1) {      // 発生条件(3)
   case 1:
      goto Label;
   default:
      break;
   }
}
-----------------------------------------------
```

コンパイル結果：

```
-----------------------------------------------
_func:
L15:
                        ; aへの代入が削除
```

```
        BRA     L15     ; 無限ループ
        NOP
        RTS
        NOP
-----------------------------------------
```

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(4)の外部変数をvolatile修飾する。

(2) optimize=0 オプションを使用する。

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 04で改修する予定です。
