RENESAS ツールニュース

RENESAS TOOL NEWS 2011年10月27日：111027/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 7 件を連絡します。

- -global_volatile=1オプションを使用する場合の注意事項 (SHC-0082)
- 局所static変数と同名の関数のアドレスを参照する場合の注意事項 (SHC-0083)
- 関数呼び出し または はvolatile修飾された変数と、 -1とのビット和 OR (|) 演算 または 0とのビット積 AND (&) 演算の注意事項 (SHC-0084)
- ループ内に、ループ制御変数を含むif文または条件演算子がある場合の注意事項 (SHC-0085)
- if文または条件演算子を含むループ中で参照している変数の値を、代入文以外の文によって更新する場合の注意事項 (SHC-0086)
- スタック渡しとなる仮引数と、関数内で確保したスタック領域の両方を、同じ制御ブロック内でアクセスする関数がある場合の注意事項（SHC-0087）
- 1つの関数内でchar型配列の一つの要素を複数回参照する場合の注意事項 (SHC-0088)

注： 各注意事項の後ろの番号は、注意事項の識別番号です。

---

1. -global_volatile=1オプションを使用する場合の注意事項 (SHC-0082)

該当バージョン：
V.9.00 Release 00 ～ V.9.04 Release 00

内容：
-global_volatile=1 オプションを使用した場合に、R0～R7またはFR0～FR11 レジスタに間違った値が格納され、プログラムが正しく動作しなくなることがあります。

発生条件：
以下の条件をすべて満たす場合に発生することがあります。
(1) -optimize=0 および -optimize=debug_only オプションを使用していない。

(2) -global_volatile=1 オプションを使用している。
(3) -noscope オプションを使用していない。
(4) (4) 最適化範囲が分割される関数が存在する。
注：（4）に該当する場合、-message オプションを使用した場合、
以下のメッセージが出力されます。
C0101 (I) Optimizing range divided in function "関数名"
"関数名"の最適化範囲が複数に分割されました。

発生例：

```
-------------------------------
. . . . .
 member[9].func(0x1000, -1, 0);
. . . . .
-------------------------------
```

正しいコンパイル結果：

```
-----------------------------------------------------------------------
 MOV       #72,R1
 SHLL2     R1
 MOV       #16,R5
 ADD       R1,R12
 MOV       #0,R7     ; func()の第4引数に0を格納
 AND       R11,R2
 MOV       #-1,R6    ; func()の第3引数に-1を格納
 SHLL8     R5        ; func()の第2引数に0x1000を格納
 LDS       R2,FPSCR
 MOV.L     L28+10,R1 ; _func__5ClassFUiiT1
 JSR       @R1
 MOV       R12,R4    ; func()の第1引数 (thisポインタ) に
                        &member[9]を格納
-----------------------------------------------------------------------
```

問題が発生した場合、R4に&member[9]を格納せず、定数値(不定値) 0x120が格納されます。これにより、func() 内において、this ポインタが間違ったアドレスを示します。

問題が発生したコンパイル結果：

```
-----------------------------------------------------------------------
 MOV       #72,R4
 MOV       #16,R5
 SHLL2     R4        ; 誤ってfunc()の第1引数 (thisポインタ) に
                       0x120が格納される
 ADD       R12,R1
 MOV       #0,R7     ; func()の第4引数に0を格納
 AND       R11,R2
 MOV       #-1,R6    ; func()の第3引数に-1を格納
```

```
    SHLL8       R5          ; func()の第2引数に0x1000を格納
    MOV.L       L28+8,R1    ; _func__5ClassFUiiT1
    JSR         @R1
    LDS         R2,FPSCR
  ----------------------------------------------------------------------
```

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 または -optimize=debug_only オプションを使用する
(2) -global_volatile=0 オプションを使用する
(3) -noscope オプションを使用する
(4) -messageオプションを指定してもメッセージC0101(I)が出力されなくなるまで、発生条件(4)に該当する関数を分割する。

2. 局所static変数と同名の関数のアドレスを参照する場合の注意事項 (SHC-0083)

該当バージョン：
V.9.00 Release 00 ～ V.9.04 Release 00

内容：
同一ファイル内で、関数内で宣言された static 変数の参照があり、かつ
それより後で、その変数と同名の関数のアドレスを参照した場合、アドレスが
関数のアドレスでなく、変数のアドレスとなる場合があります。

発生条件:
以下の条件をすべて満たす場合に発生することがあります。
(1) 関数内で宣言されたstatic変数を参照している関数がある。
(2) (1)の関数が定義されているファイル内にある(1)と別の関数内に、
(1)のstatic変数と同名の関数のアドレス参照、または関数呼び出しがある。
(3) (2)の関数アドレス参照または関数呼び出しは、(1)のstatic変数の参照より後にある。
注： (1)の関数と(2)の関数が一致する場合も該当する。

発生例：

```
 ------------------------------------------------
extern void clock(void);
void foo(void)
 {
   static int clock = 1; // 発生条件(1)
   clock++;
 }
void foo2(void)
 {
```

```
  clock();      // 発生条件(2)および(3)
}
-----------------------------------------------
```

コンパイル結果：

```
-------------------------------------------------------------------
 _foo2:
    MOV.L      L12,R2   ; __$clock$2   <- 誤ってstatic変数を参照する
    JMP        @R2
    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(1)に該当するstatic変数と、発生条件(2)に該当する
   関数を別の名前にする。

(2) 発生条件(1)の変数参照より前で、発生条件(3)の関数アドレスを
   volatile変数に代入する。

(3) 発生条件(3)を満たさないように、関数の順序を入れ替える。
   発生例の回避例： foo()とfoo2()を入れ替える。

回避策(2)の場合の回避例：

```
-----------------------------------------
extern void clock(void);

//// 追加 ////
void dummy(){
   volatile FUN_t f = &clock;  // 発生条件(3)の関数アドレスをvolatile
                               変数に代入
   f();
}
//// 追加ここまで ////

void foo(void)
{
   static int clock = 1; // 発生条件(1)
   clock++;
}

void foo2(void)
{
   clock();      // 発生条件(2)および(3)
}
-----------------------------------------
```

3. 関数呼び出し または はvolatile修飾された変数と、 -1とのビット和 OR (|)

演算 または 0とのビット積 AND (&) 演算の注意事項 (SHC-0084)

該当バージョン：
V.9.02 Release 00 ～ V.9.04 Release 00

内容:
関数呼び出し、または volatile 修飾された変数をオペランドに持つ
演算式において、その演算が-1との論理和"|"、または0との論理積"&"で
ある場合に、その関数呼び出しが行われない、またはvolatile 修飾した
変数へのアクセスが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す場合に発生することがあります。
(1) 以下のいずれかの型のオペランドを持つビットごとの論理和"|"、または
ビットごとの論理積"&"演算がある。
long long, signed long long および unsigned long long
(2) (1)の演算は以下のいずれかを満たす。
(2-1) "|"の場合、-1をオペランドにもつ。
(2-2) "&"の場合、0をオペランドにもつ。
注： オペランドは、定数伝播 (const修飾された外部変数の定数
伝播を含む) の最適化により変数が定数に置き換わった場合も含む。
(3) (2)の演算のもう一方のオペランドが、以下のいずれかである。
(3-1) volatile 修飾された変数
(3-2) -global_volatile=1オプションを使用している場合、外部変数
(3-3) 関数呼び出し
(3-4) (3-1)、(3-2)および(3-3)のいずれかを含む式

発生例:

```
long long a;
int sub();
main(){
  long long x;
  x = -1LL;                    // 発生条件(2-1)
  a = ((long long)(sub()+2)) | x;  // 発生条件(1)、(2-1)、および(3-4)
}
```

上記の例をコンパイルすると、((long long)(sub()+2)) | -1LL が誤って削除
され、関数呼び出し sub() が行われません。

コンパイル結果:

```
_main:
    MOV.L     L11+2,R6   ; _a
    MOV       #-1,R2     ; H'FFFFFFFF
    MOV.L     R2,@R6     ; part of variable a
    RTS
    MOV.L     R2,@(4,R6) ; part of variable a
```

回避策：

発生条件(2)の定数 -1 または 0 を、volatile 修飾された変数に代入し
定数の代わりにその変数を使用する。

発生例の回避例：

```
long long a;
int sub();
main(){
  volatile long long x;  // 発生条件(2)の定数の変わりにvolatile修飾
                         された変数を使用
  x = -1LL;                      // 発生条件(2-1)
  a = ((long long)(sub()+2)) | x;  // 発生条件(1)および(3-4)
}
```

---

4. ループ内に、ループ制御変数を含むif文または条件演算子がある場合の注意事項

(SHC-0085)

該当バージョン：

V.9.00 Release 00 ～ V.9.04 Release 00

内容:

ループ内に、ループ制御変数を含むif文または条件演算子がある場合、
そのif文または条件演算子の判定を誤る場合があります。
注： ループ制御変数は、ループ内で繰り返しごとに値が一定に増加または
減少し、かつループを繰り返すかどうかの判定式で参照される変数です。

発生条件:

以下の条件をすべて満たす場合に発生することがあります。
(1) -optimize=0 および -optimize=debug_only オプションを使用して
いない。
(2) ループ制御変数を持つループが存在する。
(3) (2)のループ制御変数の初期値および上限値はいずれも定数である。
注： 定数は、定数伝播 (const修飾された外部変数の定数伝播を含む) の

最適化により変数が定数に置き換わった場合も含む。
(4) (2)のループ内にif文または条件演算子「?:」がある。
(5) (4)のif文または条件演算子の制御式は、以下をすべて満たす。
(5-1) 大小比較演算子「<」「>」「<=」または「>=」による比較式である。
(5-2) 一方のオペランドは、(2)のループ制御変数である。
(5-3) もう一方のオペランドは、ループ制御変数の上限値以下の整数
定数である。
注： 整数定数は、定数伝播 (const修飾された外部変数の定数
伝播を含む) の最適化により変数が定数に置き換わった
場合も含む。

発生例:

```
------------------------------------------------------------
 int main(void) {
  int  j;
  char a1[6],a2[6],a3[6];
  for ( j = 0; j < 6; j++ ){  // 発生条件(2)、(3)および(4)
   if ((j >= 0 && j < 2) || (j >= 4 && j < 6))  // 発生条件(5)
    a1[j] = 1;
   else
    a1[j] = 0;
   if (j < 4)    // 発生条件(5)
    a2[j] = 2;
   else
    a2[j] = 0;
   if (j >= 0 && j < 1)    // 発生条件(5)
     a3[j] = 3;
   else

    a3[j] = 0;
  }
 }
------------------------------------------------------------
```

上記の発生例をオプション -cpu=sh4a -speed を使用してコンパイルすると
問題が発生します。
なお、本例では、発生条件(5)に該当する一つ目の制御式「j < 2」でのみ
問題が発生します。

コンパイル結果：

```
------------------------------------------------
    MOV       #0,R6     ; H'00000000
    MOV       #2,R1     ; H'00000002
    MOV       R15,R4
    MOV       R15,R5
    ADD       #8,R4
```

```
    MOV       R1,R13
    ADD       #16,R5
    MOV       R6,R7
    MOV       R15,R2
    MOV       #1,R12    ; H'00000001
    MOV       #3,R9     ; H'00000003
    MOV       #6,R14    ; H'00000006
    MOV       #4,R10    ; H'00000004
L11:
    CMP/PZ    R6
    MOV.B     R12,@R4   ; a1[] . . . (A) 誤ってa1[3] に1が設定される
    BF/S      L12
    MOV.B     R13,@R2   ; a2[]
    CMP/GE    R12,R6
    BT        L12
    MOV.B     R9,@R5    ; a3[]
L15:
    ADD       #1,R6
    CMP/GE    R14,R6
    ADD       #1,R5
    ADD       #1,R4
    BT/S      L17
    ADD       #1,R2
    DT        R1        ; (B) j(R6) が 3 の時、R1が-1になる
    BF        L11       ; (C) 誤ってL11 に分岐
    CMP/GE    R10,R6
    BT        L20
    MOV.B     R7,@R4    ; a1[] . . . (D) 正しくはここでa1[3] に
                            0が設定される
    BRA       L12
    MOV.B     R13,@R2   ; a2[]
L20:
    CMP/GE    R14,R6
    BF        L22
    BRA       L23
    MOV.B     R7,@R4    ; a1[]
L22:
    MOV.B     R12,@R4   ; a1[]
L23:
    MOV.B     R7,@R2    ; a2[]
L12:
    BRA       L15
    MOV.B     R7,@R5    ; a3[]
L17:
. . . . . .
```

上記のコンパイル結果において、j(R6) が 3 のとき、(B)のR1 は -1になり
ます。この結果、(C)で L11 に分岐し、(A)で a1[3] に 1 が設定され、
誤った結果になります。
正しく動作した場合は、(C)で分岐せず、(D)でa1[3] に 0 が設定されます。

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 または debug_only を使用する。
(2) 発生条件(2)のループ制御変数を volatile 修飾する。

5. if文または条件演算子を含むループ中で参照している変数の値を、代入文以外の文によって更新する場合の注意事項 (SHC-0086)

該当バージョン：
V.9.04 Release 00

内容:
if文または条件演算子を含むループがあり、そのループ内で参照している
外部変数またはstatic変数の値を、この変数への代入文以外の文によって
更新する場合、ループ抜け出し後に更新前の値に戻る場合があります。

発生条件:
以下の条件をすべて満たす場合に発生することがあります。
(1) -optimize=0 および -optimize=debug_only オプションを使用して
いない。
(2) -noscope オプションを使用していない。
(3) 最適化範囲が分割される関数が存在する。
注：（3）に該当する場合、-message オプションを使用した場合、
以下のメッセージが出力されます。
C0101 (I) Optimizing range divided in function "関数名"
"関数名"の最適化範囲が複数に分割されました。

注： 以降、C0101 (I)で示された関数を、発生条件(3)の関数と呼ぶ

(4) (3)の関数内にif文または条件演算子「?:」を含むループが存在する。
(5) (4)のループは内側に別のループを含まない、かつ、無限ループでない。
(6) 以下をすべて満たす外部変数またはstatic変数がある。
(6-1) (4)のループ内で定義および参照されない
(6-2) (3)の関数内で定義または参照されている
(7) (6)の変数はvolatile修飾されていない。
かつ -global_volatile=1オプションを使用していない。
(8) (6)の変数は(4)のループ内で、この変数への代入文以外の文によって

更新される。(例：関数呼び出し)

発生例：

```
 ---------------------------------------------------
 int aaa,xxx=0,yyy=0,n;       // 発生条件(7)
 void sub(void);
 void func(void)             // 発生条件(3)
 {
  . . . . . . . . . . . . . . . . . . .
   aaa = 0;                  // 発生条件(6)
  . . . . . . . . . . . . . . . . . . .
   for (i = 0; i < n ;i++) {   // 発生条件(4)および(5)
     if(xxx == yyy){
       sub();                // 発生条件(8)
     }
   }
  . . . . . . . . . . . . . . . . . . .
 }
 void sub(void)
 {
   aaa++;
 }
 ---------------------------------------------------
```

コンパイル結果：

```
 ---------------------------------------------------
 _func:
 . . . . . . . . . . . .
     MOV.L      @R9,R13    ; aaa . . . (A)
     MOV.L      L17+8,R12  ; _n
     MOV.L      L17+12,R10 ; _yyy
     MOV.L      L17+16,R11 ; _xxx
     BRA        L11
     MOV        #0,R14     ; H'00000000
 L12:
     MOV.L      @R10,R6    ; yyy
     MOV.L      @R11,R2    ; xxx
     CMP/EQ     R6,R2
     BF         L14
     BSR        _sub       ; aaa++; . . . (B)
     NOP
 L14:
     ADD        #1,R14
 L11:
     MOV.L      @R12,R2    ; n
```

```
    CMP/GE     R2,R14
    BF         L12       ; if (i < n)
    MOV.L      R13,@R9   ; aaa . . . (C)
 . . . . . .
_sub:
    MOV.L      L17,R6    ; _aaa
    MOV.L      @R6,R2    ; aaa
    ADD        #1,R2
    RTS
    MOV.L      R2,@R6    ; aaa
```

上記のコンパイル結果において、(A)でaaaからR13にロードした値を
(C)でaaaからR13に書き戻すため、
(B)で変更されたaaaの値が、ループ抜け出し後にループ直前の値に戻ります。

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 または -optimize=debug_only オプションを使用する。
(2) -noscope オプションを使用する。
(3) -global_volatile=1 オプションを使用する。
(4) 発生条件(6)の変数をvolatile修飾する。
(5) 発生条件(4)のループ内で発生条件(6)の変数を参照する。
(6) -messageオプションを指定してもメッセージC0101(I)が出力されなくなる
まで、発生条件(3)に該当する関数を分割する。

6. スタック渡しとなる仮引数と、関数内で確保したスタック領域の両方を、同じ

制御ブロック内でアクセスする関数がある場合の注意事項（SHC-0087）

該当バージョン：
V.9.00 Release 00 ～ V.9.04 Release 00

内容:
スタック渡しとなる仮引数と、関数内で確保したスタック領域の両方を、
同じ制御ブロック内でアクセスする関数がある場合、そのアクセスの結果が
正しくないことがあります。

発生条件:
以下の条件をすべて満たす場合に発生することがあります。
(1) -optimize=0 および -optimize=debug_only オプションを使用して
いない。
(2) -cpu=sh2afpu, -cpu=sh4, または -cpu=sh4a オプションを使用している。
(3) -fpu=single オプションを使用していない。
(4) スタック渡しの仮引数を持つ関数がある。

(5) (4)の仮引数の型は以下のいずれかであり、かつ volatile 修飾されていない。
(5-1) double型
(5-2) long double型
(5-3) double型 または long double型のメンバを持つ構造体型または共用体型
(6) (4)の関数内で、レジスタ退避および回復以外に、スタック領域を確保している。
コンパイルリスト中で、(4)の関数の.STACK制御命令の値が、関数先頭でレジスタを退避するのに使用したスタック領域のサイズよりも大きい場合が該当です。
(7) (6)のスタック領域に割り付けられた、float 型、double 型、または long double 型の変数をアクセスしている。
(8) (5)の仮引数を参照している。
ただし、(5-3)に該当する場合は、その double 型 または long double 型メンバをアクセスする場合のみ該当。
(9) (7)のスタック領域アクセスと、(8)の参照の間に、if文、switch文、for文、while文、do-while文、goto文、および条件演算子がない。

発生例:
オプション -cpu=sh2afpu を使用する場合

```
struct tbl {
  double x[2];
};
void sub(float a, double b);
void func(struct tbl s){     // 発生条件(4)および(5-3)
  float y[4];                // 発生条件(6)
  sub(y[3],s.x[1]);    // 発生条件(7)、(8)および(9)
}
```

コンパイル結果：

```
_func:
    .STACK      _func=16              ; 発生条件(6)
    ADD         #-16,R15
    FMOV.S      @(24,R15),FR6
    MOV         #28,R0     ; H'0000001C
    FMOV.S      @(R0,R15),FR7; s.x[]
    FMOV.S      FR7,FR4   ; (A) 誤ってs.x[1]の下位 4 バイトを転送する
    MOV.L       L11+2,R2   ; _sub
    JMP         @R2
    ADD         #16,R15
```

上記のコンパイル結果において、(A)で y[3] を FR4 に転送せず、
s.x[1]の下位 4 バイトを転送します。

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 または debug_only を使用する。
(2) -fpu=single を使用する。
(3) 発生条件(4)の仮引数を volatile 修飾する

発生例の回避例：

```
-----------------------------------------------
  void func(volatile struct tbl s){
-----------------------------------------------
```

7. 1つの関数内でchar型配列の一つの要素を複数回参照する場合の注意事項

（SHC-0088）

該当バージョン：
V.9.04 Release 00

内容:
1つの関数内にchar、signed char、または unsigned char 型の同じ配列
要素の参照が複数ある場合、配列要素に正しくアクセスできないことがあります。

発生条件:
以下の発生条件Aまたは発生条件Bを満たす場合に発生することがあります。

発生条件A：
以下の(A1)～(A6)をすべて満たす場合に発生することがあります。
(A1) -optimize=0 および -optimize=debug_only オプションを使用して
いない。
(A2) char、signed char、または unsigned char 型の配列、または
それらの型を指すポインタ変数がある。
(A2) char、signed char、unsigned char 型の配列があるか、または
それらの型を指すポインタ変数が宣言されている。
(A3) (A2)の配列の要素が、1つの関数内で2回以上参照される。
なお要素は、配列参照と等価な間接参照式を記述した場合も含む。
(間接参照式の例： *(a+exp)はa[exp]と等価)
(A4) (A3)の配列要素の添え字式は、加算式または減算式であり、かつ
その一方のオペランドは変数で、かつもう一方のオペランドは0以外の
定数である。
(A3)の間接参照式の例の場合、expが添え字式に相当します。
注： 定数は、定数伝播 (const修飾された外部変数の定数伝播を含む)

の最適化により変数が定数に置き換わった場合も含む。
(A5) (A4)の変数は、char、signed char、unsigned char、short、signed short、または unsigned short 型である。
(A6) (A4)の添え字式の値が、(A5)の変数の型で表現可能な最大値より大きい値になる。

発生条件Aの発生例:

```
---------------------------------------------------------------
char S,*ary;                // 発生条件(A2)
void func(char par)         // 発生条件(A5)
    {
      if (ary[par+1] > 10) {        // 発生条件(A3)および(A4)
        S = ary[par+1];             // 発生条件(A3)および(A4)
      }
      return;
    }
---------------------------------------------------------------
```

上記の発生例では、par が 127 の場合に、発生条件(A6)に該当します。

コンパイル結果：

```
---------------------------------------------------------------
_func:
   MOV.L      L13,R7     ; _ary
   ADD        #1,R4
   MOV.L      @R7,R0     ; ary
   EXTS.B     R4,R1      ; 発生例のpar+1を誤って符号拡張する
   MOV.B      @(R0,R1),R6; ary[]
   MOV        #10,R5     ; H'0000000A
   CMP/GT     R5,R6
   BF         L12
   MOV.L      L13+4,R2   ; _S
   MOV.B      R6,@R2     ; S
---------------------------------------------------------------
```

発生条件B：

以下の(B1)～(B6)をすべて満たす場合に発生することがあります。
(B1) -optimize=0 および -optimize=debug_only オプションを使用していない。
(B2) char、signed char、または unsigned char 型の配列メンバを持つ構造体または共用体が存在する。
(B3) (B2)の配列メンバの同じ要素を、1つの関数内で2回以上参照する。
(B4) (B3)の配列要素の添え字式は、以下のいずれかである。
(B4-1) 変数
(B4-2) 加算式で、かつ一方のオペランドが変数で、もう一方のオペランドが定数

(B4-3) 減算式で、かつ一方のオペランドが変数で、もう一方の
オペランドが定数
注： (B4-2)と(B4-3)の定数は、定数伝播 (const修飾された外部変数の
定数伝播を含む) の最適化により変数が定数に置き換わった
場合も含む。
(B5) (B4)の変数は、char、signed char、unsigned char、short、
signed short、または unsigned short 型である。
(B6) (B3)の配列要素の、(B2)の構造体または共用体先頭からのオフセット
が、(B4)の変数の型の表現可能な最大値より大きい値となる。

発生条件Bの発生例:

```
----------------------------------------------------------
char S;
struct {
  char dummy[127];
  char mem[8];                  // 発生条件(B6)
} *st;                        // 発生条件(B2)
void func(char par)           // 発生条件(B5)
{
  if (st->mem[par] > 10) {      // 発生条件(B3)および(B4-1)
    S = st->mem[par];           // 発生条件(B3)および(B4-1)
  }
  return;
}
----------------------------------------------------------
```

上記の発生例では、par が 1 以上 の場合に、発生条件(B6)に該当します。

コンパイル結果：

```
----------------------------------------------------------
_func:
      MOV.L      L13,R7     ; _st
      ADD        #127,R4
      MOV.L      @R7,R0     ; st
      EXTS.B     R4,R1      ; 発生例のpar+127を誤って符号拡張する
      MOV.B      @(R0,R1),R6; st->mem[]
      MOV        #10,R5     ; H'0000000A
      CMP/GT     R5,R6
     BF         L12
      MOV.L      L13+4,R2   ; _S
      MOV.B      R6,@R2     ; S
----------------------------------------------------------
```

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 または debug_only オプションを使用する。

(2) 以下のいずれかをvolatile修飾する。
- 発生条件(A2)の配列
- 発生条件(B2)の配列メンバ
- 発生条件(B2)の構造体または共用体
- 発生条件(A4)または(B4)の添え字式中の変数

(3) 発生条件(A4)または(B4)の添え字式中の変数を、発生条件(A5)または(B5)以外の型に変更する。

発生例Bの回避例：

```
------------------------------------------------------------
    void func(signed int par)   // char をsigned int に変更
------------------------------------------------------------
```

(4) 発生条件(B4-1)を満たす場合、添え字式中の変数を発生条件(B5)以外の型にキャストする。

回避例：

```
------------------------------------------------------------
    st->mem[(signed int)par]   // par を signed int にキャスト
------------------------------------------------------------
```

8. 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.9.04 Release 01 で改修しました。
V.9.04 Release 01の詳細は、RENESAS TOOL NEWS 資料番号 111027/tn2 を参照ください。以下のURLで参照できます。(10月27日から公開予定)
https://www.renesas.com/search/keyword-search.html#genre=document&q=111027tn2