SuperH RISC engine C/C++コンパイラ Ver.8 使用上のご注意(2)

SHC コンパイラ Ver.8 には、以下に記載しました不具合があります。 SHC コンパイラ Ver.8 をご使用の場合には、ご注意いただきますよう、お願いいたします。
なお、本ページに記載されている不具合は、コンパイラ V.8.00.02 以降（コンパイラパッケージ V.8.00 Release02 以降）では、全て修正されています。製品をリビジョンアップしていただきますよう、お願いいたします。

該当製品

製品型名	パッケージバージョン	コンパイラバージョン
P0700CAS8-MWR	8.0.00, 8.0.01	8.0.00, 8.0.01
P0700CAS8-SLR	8.0.00, 8.0.01	8.0.00, 8.0.01
P0700CAS8-H7R	8.0.00, 8.0.01	8.0.00, 8.0.01

1. コピー伝播不正

現象

複数の分岐元を持つブロックにコピー命令が存在した場合、不正にコピー命令を削除する場合があります。

```
[例]
int func(int *x) {
    int ret=0;
    while(*x++){
       if(*x==1){
          ret+=2;
        }
     }
     return (ret+2);
}
    _func:
          MOV          #0,R5   ; 不正にコピーを削除した結果 R7 -> R5 へ変換
      L11:
          MOV.L       @R4,R2
          ADD          #4,R4
                                ; *1 MOV R7,R5 を不正に削除
          TST          R2,R2
          ADD          #2,R5
          BT           L13
          MOV.L       @R4,R0
          CMP/EQ      #1,R0
          BT           L11      ; *2 *3 により BF L11 を変換
          BRA          L11
          NOP                   ; *3 MOV R5,R7 を不正に削除
      L13:
          RTS
          MOV          R5,R0
```

発生条件

次の条件を全て満たす時、発生することがあります。

(1) optimize=1 を指定している。
(2) 条件文を記述している。
(3) 複数の分岐元を持つブロックにコピー命令が存在する(例では*1)。
(4) (3)の分岐元のブロックにコピー命令のコピー元レジスタ(例では R7)の定義がないパスがある。
　(例では*2 から L11 へ分岐するパス)。

回避策

該当箇所が存在した場合、以下の方法で回避してください。

・optimize=0 を指定する。

2.不要式削除不正

現象

条件文の then 節、あるいは else 節に代入式があり、その直後に同じ変数同士の代入式を記述した場合、不正に条件文を削除する場合があります。

[例]
```
int x;


void f(int y){
   if (y>=256){ /* 不正に削除    */
       x=0;     /* *1    */
   }
   x=x;         /* *2   同じ変数同士の代入文削除 */
   x++;
}
```

↓

```
void f(int y){
   x=0;
   x++; /* x=0 を伝播 */
}
```

↓

```
void f(int y){
   x=1;
}
```

発生条件

次の条件を全て満たす時、発生することがあります。

(1) optimize=1 を指定している。

(2) 条件文を記述している。
(3) (2)の条件文の then 節もしくは else 節に代入式がある(例では*1)。
(4) (2)の条件文の後に(3)の代入先変数同士の代入式がある(例では*2)。

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。

(1) optimize=0 を指定する。
(2) opt_range=noblock を指定する。

3. スタック渡し引数アクセス不正

現象

スタック渡し引数を持つ関数の出口直前に関数呼び出しがある場合、speed オプションを指定するとスタック渡し引数を参照するアドレスが不正になる場合があります。

```
[例]
typedef struct {
    int x;
} ST;
extern void g(ST *x);
void f(int a, ST b) {   /* b はスタック渡し引数 */
   if (a) {
       g(&b);
      /* (A) */
   }
   /* (B) */
}

; 関数入口での引数 b の格納アドレス = R15
   f:
            TST         R4,R4
            BT          L12
            MOV         R15,R4
            MOV.L       L14,R2      ; _g
            JMP         @R2         ; (A)
            ADD         #4,R4       ; R4 <- R15+4 : b のアドレスではない
   L12:
            RTS                     ; (B)
            NOP

        ↓

void f(int y){
   x=1;
}
```

発生条件

次の条件を全て満たす時、発生することがあります。

(1) optimize=1 を指定している。

(2) speed オプションを指定している。
(3) 当該関数がスタック渡し引数を持つ(例では b)。
(4) 当該関数の関数出口が複数ある(例では(A),(B)の 2 ヵ所)。
(5) (4)の中で直前が関数呼び出しである出口がある(例では g(&b);)。
(6) 当該関数内での関数呼び出しは(5)のみである。

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。

(1) speed オプション指定をしない。
(2) optimize=0 を指定する。
(3) 関数呼び出し後に nop()組み込み関数を挿入する。
(4) 当該関数内にダミーの関数呼び出しを挿入し、noinline オプションを指定する。

4. GBR 相対論理演算不正

現象

#pragma gbr_base/gbr_base1 を指定した 1byte の配列、もしくはビットフィールドメンバに対し論理演算を行った場合、論理演算の結果を不正な領域に書き込む場合があります。

[例]
```
#pragma gbr_base a,b
    char a[2],b[2];
    void f() {
        a[0] = b[0] & 1;
    }

        MOV         #_b-(STARTOF $G0),R0
        RTS
        AND.B       #1,@(R0,GBR)                ; 演算結果を b[0]に書き込み
```

発生条件

次の条件を全て満たす時、発生することがあります。

(1) gbr=user を指定している。
(2) #pragma gbr_base/gbr_base1 で以下の変数を指定している。
  ・(unsigned)char 型の配列
  ・(unsigned)char 型のメンバを持つ構造体配列
  ・(unsigned)char 型の配列メンバを持つ構造体
  ・8bit 以下のビットフィールドメンバを持つ構造体
(3) (2)の変数(例では b[0])に対して定数との論理演算(&, |, ^)を行っている。
(4) (3)の演算の代入先変数(例では a[0])が(2)の条件を満たす。
(5) (3)、(4)の変数は別変数、同じ配列で別要素、または同じ構造体で別メンバである。

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。

(1) #pragma gbr_base/gbr_base1 指定をやめる。

(2) gbr=auto を指定する(Warning を出力し、#pragma gbr_base/gbr_base1 指定を無効にする)。

(3) 論理演算の結果をいったん volatile 指定したテンポラリ変数に代入する。

例

```
  void f() {
    volatile char temp;
    temp = b[0] & 1;
    a[0] = temp;
  }
```

5. 符号拡張削除不正

現象

変数、定数アドレスや配列のインデックスを 1,2byte にキャストした後に、その値を用いてメモリアクセスを行った場合、または unsigned short 型の変数に、char 型にキャストした式を代入してから比較式で使用した場合、不正にキャストが削除されることがあります。

[例 1]

```
unsigned short x;
char a[1000];

void f() {
   a[(char)x] = 0;
}
   MOV.L      L11+2,R2         ; _x
   MOV.L      L11+6,R6         ; _a
   MOV.W      @R2,R5
   EXTU.B     R5,R0
                               ; EXTS.B R0,R0 を削除
   MOV        #0,R5            ; H'00000000
   RTS
   MOV.B      R5,@(R0,R6)      ; x が 0～127 の範囲外の時、不正アドレスを
                               ; 参照する場合あり
```

[例 2]

```
unsigned short sc0;
unsigned int b;


func1() {
    unsigned short us1;
    us1 = (char)b ;
    return(us0 !=us1);
}

    MOV.L       L11,R2       ; _b
    MOV.L       L11+4,R5     ; _us0
    MOV.L       @R2,R6
    EXTS.B      R6,R2
    MOV.W       @R5,R6
```

```
EXTU.W      R6,R5
CMP/EQ      R2,R5       ; (char)b を unsigned short にキャストしないまま比較
MOVT        R0
RTS
XOR         #1,R0
```

発生条件

次の条件を全て満たす時、発生することがあります。

(1) optimize=1 を指定している。
(2) 以下の(a)(b)のいずれかの条件を満たす。
  (a)
    (a-1) 変数アドレス、定数アドレス、配列のインデックスを明示的に 1,2byte にキャストしている、もしくは当該関数が char/short の仮引数を持ち、その引数を配列のインデックスのみで使用している。
    (a-2) (a-1)の値を用いてメモリアクセスを行う。
  (b)
    (b-1) unsigned short 型の変数に、char 型にキャストした式を代入している。
    (b-2) (b-1)の変数を比較式で使用している。

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。

(1) optimize=0 を指定する。
(2) 発生条件(2)(b)が成立した場合、条件に該当する unsigned short 型の変数を volatile 修飾する。

株式会社 日立超LSIシステムズ