SH C/C++コンパイラ V.8.00.02、V.8.00.03 で対策した制限事項

以下の 12 点の制限事項を対策しました。

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る。

```
[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

以下のすべての条件を満たした場合に発生することがある。

(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)。

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る。

```
[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

以下のすべての条件を満たした場合に発生することがある。

(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)。

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る。

```
[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
```

発生条件

以下のすべての条件を満たした場合に発生することがある。

(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。

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る。

[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

以下のすべての条件を満たした場合に発生することがある。

(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。

5. 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、キャストが削除され不正なメモリ領域をアクセスする場合がある。

[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 us0;
  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

以下のすべての条件を満たした場合に発生することがある。

(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。

6. 制限事項

外部変数定義の初期値にセクションアドレス演算子と数値の加減算を記述した場合、オブジェクト出力において、数値の加減算の部分が出力されない。

[例]

```
  char *addr1 = (char *)__sectop("P");                    // OK
  char *addr2 = (char *)__sectop("P") + __secsize("P"); // OK
  char *addr3 = (char *)__sectop("P") + 10;               // NG
```

7. インターナルエラー

以下の場合、インターナルエラーが発生する。

(1) 関数型の typedef を使用した static 関数メンバをもつクラスの宣言のみ存在する C++プログラムを debug オプションを指定してコンパイルした場合。

(2) {}で囲まれてなく、かつ default ラベルのない switch 文を持つ C/C++プログラムを optimize=0 かつ debug オプションを指定してコンパイルした場合。

8. ゼロ拡張削除不正

現象

ループ内で 2 回以上参照する unsigned char/unsigned short 型変数のゼロ拡張が不正に削除される場合がある。

[例]

```
        MOV.B    @Rm,Rn
        EXTU.B   Rn,  Rn      ;上位 3byte をクリア
                 :
        MOV.B    Rn,@R15      ;スタックに割付
                 :
        MOV.B    @R15,R12     ;R12 に割付 => EXTU.B R12,R12 が出ない
  L1:
```

```
        :
CMP/EQ  R12,R2      ;R12 の値が不正
        :
BT      L1
        :
```

発生条件

以下のすべての条件を満たした場合に発生することがある。

(1) optimize=1 を指定している。

(2) unsigned char/unsigned short 型変数が存在する。

(3) (2)の変数がループ内を含めて 2 回以上参照される。

(4) (2)の変数がレジスタに割り付かない。

(5) (3)のループ内で使用されないレジスタが存在する。

(6) (5)のレジスタがループ外で使用されている。

9. 加算/減算/乗算で拡張削除不正

現象

加算/減算/乗算の結果を、それより小さなサイズの型の変数に代入、もしくは小さなサイズの型に変換し、その結果を加算/減算/乗算で使った場合、拡張命令が不正に削除される場合がある。

[例]

```
  int x,a;
  test_OOO() {
      char b;
      b = (char)(a + 3);
      x = b + 2;
  }

  _f:
      MOV.L   L11,R6      ; _a
      MOV.L   L11+4,R2    ; _x
      MOV.L   @R6,R6
      ADD     #5,R6       ; char へのキャストが削除され、
                            a+5 の結果を x に代入している。
      RTS
      MOV.L   R6,@R2
```

発生条件

以下のすべての条件を満たした場合に発生することがある。

(1) optimize=1 を指定している。

(2) ２つのオペランドのうち片方のみが定数である加算、減算、乗算が存在する。

(3) (a),(b)のいずれかが成立する。

  (a) (2)の結果を、それより小さなサイズの型にキャストし、その結果を加算、減算、乗算している。

(b) (2)の結果を、それより小さなサイズの型の変数に代入し、その結果を加算、減算、乗算している。

10. ループ変数の 2 次式の演算不正

現象

ループ内にループ変数の 2 次式が存在する場合、その演算結果が不正になることがある。

```
[例]
  int a[100];
  void f() {
    int i;
    for (i=0;i<100;i++) {
      a[i] = 3 * (i * i + 555 * i); /* 不正に 3*i*i+555*i に展開 */
    }
  }
```

発生条件

以下のすべての条件を満たした場合に発生することがある。

(1) optimize=1 を指定している。

(2) ループが存在する。

(3) (2)のループ変数が int/unsigned int/long/unsigned long 型である。

(4) (2)のループ内に(3)のループ変数の 2 次式が存在する。

(5) (4)は"m*(i*i+b*i)"(i：ループ変数、m,b：変数もしくは定数)の形である。

11. 定数除算の拡張削除不正 (SHC-0001)

現象

定数除算において、除数と被除数をともにより小さなサイズの型に変換し、かつ除算結果を変換した型の変数に代入した場合、型変換が不正に削除される場合がある。

```
[例]
  char c;
  int  i;
  void func1() {
    c = ((char)i / (char)2);      /* 被除数を int 型のまま計算 */
  }

  void func2(){
    c = ((char)i / (char)0x102); /* 除数を 0x102 のまま計算    */
  }
```

発生条件

以下のすべての条件を満たした場合に発生することがある。

(1) optimize=1 を指定している。

(2) 定数除算が存在する。

(3) (2)の除算で、除数と被除数をともにより小さなサイズの型へ変換している。

(4) cpu=sh1 以外かつ division=cpu=inline を指定している。
もしくは除数の値が 2 のべき乗である。

(5) 除算結果を(3)の変換後の型の変数に代入している。

12. ループ変数の置換不正 (SHC-0003)

現象

ループに型の異なるループ変数が存在する場合に、不正にループ変数が共通化される場合がある。

[例]

```
    extern void g();
    void func(unsigned int x) {
        unsigned long i=3;
        signed long k=3;

        while (i<x) {
            if (k<-3) {  /* 不正に k を i に置換 */
                break;
            }
            g();
            --i;
            --k;
        }
    }
```

発生条件

以下のすべての条件を満たした場合に発生することがある。

(1) optimize=1 を指定している。

(2) ループが存在する。

(3) (2)のループ内に、signed int 型もしくは signed long 型のループ変数と、unsigned int 型もしくは unsigned long 型のループ変数が存在する。

(4) (3)のループ変数の初期値が共に定数である。

(5) (3)のループ変数の更新値が同じである。