SuperH RISC engine C/C++コンパイラ Ver.7.0B 不具合情報

- 過去のお知らせ

SuperH RISC engine C/C++コンパイラ Ver.7.0B に下記に示す 9 件の不具合がございましたので慎んでご連絡いたします。
ご使用の際は不具合の内容をご理解いただき、下記に示す回避策を講じていただきますよう、よろしくお願い申し上げます。
なお、下記の項番 1～8 の不具合につきましては、コンパイラの次期リビジョンアップで対策いたします。Windows 版につきましては、アップデートプログラムを 2 月中旬に弊社ホームページに掲載する予定です。項番 9 の不具合に関しましては制限事項とさせていただきます。今後ともよろしくお願い申し上げます。

1. 符号拡張/ゼロ拡張不正

現象：
関数内に結果が同じ整数型の部分式が存在する場合、その結果の下位 2 バイトまたは 1 バイトを不正に符号拡張/ゼロ拡張する場合があります。

```
[例]
  short sub(long);
  short a,b,c,d;

  void f()
  {
      if(d) {
          b = sub((long)a * (long)b * (long)c / 0x4000);
                /* ^^^^^^^^^^^^^^^^^ */
      } else {
          b = sub((long)a * (long)b * / 0x80);
                /* ^^^^^^^^^^^^^^^^ */
      }
  }
```

発生条件：
次の条件を全て満たす時、発生します。
(a) 関数内に結果が同じである共通な部分式がある(上記例の^^^^^)。
(b) その部分式の変数の値が 2 バイトまたは 1 バイト以下であることが保証できる。
(上記例では short 型の変数を型変換しているので、値としては short 型の範囲内であることが保証できる)

回避策：
次のいずれかの方法で回避できます。
(a) optimize=0 を指定する。
(b) 下記回避例のように +0 を部分式に挿入する。

```
[例]
  f() {
```

```
    if(d) {

        b = sub(((long)a+0) * ((long)b+0) * (long)c / 0x4000);
             /* ^        ^^^ ^ ^        ^^^ 追加 */
    } else {
        b = sub(((long)a+0) * ((long)b+0) * / 0x80);
             /* ^        ^^^ ^ ^        ^^^ 追加 */
    }
}
```

2. レジスタ破壊不正

現象：

optimize=1 指定時、使用中のレジスタを不正に破壊する場合があります。

発生条件：

次の条件を全て満たす時、発生します。

(a) optimize=1 を指定する。

(b) 関数内に分岐先(ループ文、return 文も含む)が同一になる複数の基本ブロック(分岐、ラベルのない命令列)が存在する。

(c) (b)の複数基本ブロックの中に同じ変数への代入文がある。

回避策：

optimize=0 を指定する。

3. １ビットフィールドメンバ定数代入不正

現象：

同じ構造体の１ビットフィールドメンバへの定数代入文が連続し、かつ同じビットフィールドメンバへの代入文が複数存在する時、インターナルエラーもしくはオブジェクト不正になる場合があります。

[例]

```
struct {
    unsigned char b1:1;
    unsigned char b2:1;
} ST;

/* インターナルエラー */
void f() {
    ST.b1=1;
    ST.b2=0;
    ST.b2=0;
}

/* オブジェクト不正 */
void g() {
    ST.b1=1;
    ST.b2=1;
    ST.b2=0;
}
```

発生条件：

次の条件を全て満たす時、発生します。

(a) 同じ構造体の 1 ビットフィールドメンバへの定数代入文が連続している。

(b) 同じビットフィールドメンバに同じ値を代入している(インターナルエラー)、もしくはビットフィールドメンバに 0 以外の定数を代入し、直後に同じビットフィールドメンバに 0 を代入している。

回避策：

次のいずれかの方法で回避できます。

(a) 構造体を volatile 宣言する。

(b) 同じメンバへの連続代入を避ける。

4. レジスタ退避・回復不正

現象：

割り込み関数内で、R0,R1,R3 が使用されているにも関わらず退避・回復されない場合があります。

発生条件：

次の条件を全て満たす時、発生します。

(a)当該関数が割り込み関数である。

(b) 当該関数内で関数呼び出しがない(実行時ルーチンを含む)。

(c) 当該関数内に以下のいずれかを含む。

- switch 文
- #pragma inline_asm 指定関数(サイズが 255 以上もしくは指定なし)の呼び出し
- 関数内分岐が 8bit で届かない場合

回避策：

ダミーの関数呼び出しを挿入する。

[例]

```
#pragma interrupt f
void g(){}
void f() {
      :
    g();
      :
}
```

5. R0 レジスタの不正破壊

現象：

optimize=1 指定時、使用中のレジスタ R0 を不正に破壊してしまう場合があります。

[例]

```
  ret = func(a,b,c);
  func2(d,e,f,g,h,ret,i,j);
              :
```

[出力コード]

```
  JSR         @R2
  MOV.L       @(R0,R15),R8
  MOV.B       R0,@(28,GBR)          ; <- func()の戻り値を ret に格納
  EXTU.B      R14,R6
  MOV.W       L2448+8,R0 ; H'0087   ; <- R0 を破壊
  MOV.B       @(R0,R15),R5
  MOV         R10,R4
  MOV.W       L2448+4,R0 ; H'0083   ; <- R0 に H'0083 を設定
  MOV.B       @(R0,R15),R7
  STC         GBR,R2
  ADD         #18,R2
  MOV.L       R9,@R15
  MOV.L       R2,@(8,R15)
  EXTU.B      R5,R5
  STC         GBR,R2
  ADD         #56,R2
  MOV.B       R0,@(7,R15)           ; <- R0(H'0083)を func2()の第 6 引数
  EXTU.B      R7,R7                 ;     として設定
  BSR         _func2
  MOV.L       R2,@(12,R15)
            :
```

発生条件：

次の条件を全て満たす時、発生する可能性があります。

(a) -optimize=1 を指定する。

(b) スタック渡しの仮引数が存在する。

回避策：

-optimize=0 を指定する。

6. switch 文条件式中の構造体メンバ参照不正

現象：

switch 文の条件式にネストした構造体のメンバ参照を記述した場合、メンバへのオフセットが不正となる場合があります。

[例]

```
  struct ST1 {
      int B0;
      int B1;
      int B2;
      int B3;
  };

  struct ST {
      struct ST1 A0;
      struct ST1 A1;
      struct ST1 A2;
      struct ST1 A3;
  };

  int x;

  test(struct ST *s) {
      switch ((&s->A1)->B2) {
```

```
      case 1:
          x=1;
      }
 }
```

[出力コード]

```
      MOV.L       @(8,R4),R0 ; <- @(24,R4)が正しい
      CMP/EQ      #1,R0
      BF          L15
      MOV.L       L17+2,R6    ; _x
      MOV         #1,R2       ; H'00000001
      RTS
      MOV.L       R2,@R6
 L15:
      RTS
      NOP
```

発生条件：

次の条件を全て満たす時、発生する可能性があります。

(a) ネストした構造体のメンバ参照がある。

(b) そのメンバの参照を、'.'ではなく、(&member)->という形で行っている。

(c) その構造体参照式が、switch 文の条件式にある。

回避策：

次のいずれかの方法で回避できます。

(a) switch 文の条件式を、その直前でいったん変数に代入した後、参照する。

```
    [例]
     test(struct ST *s) {
          int temp;
          temp=(&s->A1)->B2;
          switch (temp) {
          case 1:
              x=1;
          }
     }
```

(b) アドレス参照後に明示的なキャスト演算子を挿入する。

```
    [例]
    test(struct ST *s) {
            switch (((struct ST1 *)(&s-;>A1))->B2) {
            case 1:
                x=1;
            }
    }
```

7. デバッグ情報不正

現象：

構造体メンバが関数へのポインタで、その関数の型がプロトタイプ形式で宣言されている場合に、デバッグ情報が不正になります。

発生条件：

次の条件を全て満たす時、発生します。

(a) C 言語で記述されている。
(b) 関数型へのポインタである構造体または共用体のメンバが存在する。

回避策：
次のように関数型の宣言を typedef 宣言にする。

```
typedef   void(*FUNC)(int,int)
struct sample {
    FUNC  vp1;
};
```

8. インターナルエラー

現象：
次の場合、インターナルエラーが発生します。
(a) ファイル名(パスを含む)に空白を含む場合。
(b) map オプションでファイル名(パスを含む)に日本語を指定した場合。

回避策：
ファイル名、map オプションに日本語を含まないようにディレクトリ構成を変更する。

9. 組込み関数(set_cr()、set_imask())使用時の最適化不正

現象：
組込み関数(set_cr()、set_imask())を跨いで最適化を行なうことがあります。
[例]

```
  #include  <machine.h>
  #define  VALUE  0x000000F0
  extern  void  g();

  void  f() {
      set_cr(VALUE);       /* バンク切り替え */
      g();
  }
```

[出力コード]

```
_f :
      MOV.L      #16,R3
      MOV.L      L11,R2 ; R2 を設定
      EXTU.B     R3,R3
      LDC        R3,SR  ; バンク切り替え(R2 -> R2_bank)
      JMP        @R2    ; R2 を使用
      NOP
L11:
      .DAT.L     _g
```

発生条件：
次の条件を全て満たす時、発生する可能性があります。

(a) optimize=1 を指定。

(b) 組込み関数 set_cr()または set_imask()を使用。

回避策：

次のいずれかの方法で回避できます。

(a) optimize=0 を指定。

(b) set_cr()または set_imask()の直後に nop()を追加する。

[例]

```
  void  f() {
        set_cr(VALUE);  /*  バンク切り替え  */
        nop();
        g();
  }
```

株式会社 日立超LSIシステムズ