RENESAS ツールニュース

RENESAS TOOL NEWS [2003年12月16日：RSO-M3T-CC32R-031216D]

M3T-CC32R
ご使用上のお願い

M32Rファミリ用 クロスツールキット M3T-CC32Rの使用上の注意事項を連絡します。

- ポインタ型の仮引数を使った同一の間接参照が、関数をはさんで2つ以上実行されるプログラムに関する注意事項

---

1. **該当製品**
   M32Rファミリ用クロスツールキット
   M3T-CC32R V.4.00 Release 1 ～ V.4.20 Release 1

2. **内容**
   ポインタ型仮引数を持つ関数内で、このポインタ型の仮引数を使った同一の間接参照を関数呼び出しをはさんで2つ以上実行する場合に、 レベル4の最適化が有効になるオプションを指定してコンパイルすると、関数呼び出しの後の間接参照を省略して代わりに、 関数の呼び出し前の間接参照の値を使うコードを生成します。

   2.1 発生条件
   次の(1)～(8)全ての条件を満たす場合に発生します。
   (1) コンパイル時に -O4の最適化と同じ内容を含む最適化オプションを指定している。(-O4, -O5, -O6, -O7いずれかの指定が有効になっているか、-Ospace または -Otimeを単独で指定している。)
   (2) ポインタ型仮引数を持つ関数Aがある。
   (3) 関数Aの中に、次の(a)～(c)に該当する(2)のポインタ型仮引数を使った間接参照Bがある。
   (a) 配列参照
   (b) 構造体メンバ参照
   ※ポインタが指している構造体の先頭メンバは除く
   (c) ポインタ+整数で示されるアドレスの間接参照

(4) 間接参照Bを含んでいる処理ブロック(*1)の先頭から、間接参照Bが実行されるまでの間に(a)～(e)いずれの実行も存在しない。

(a) (2)のポインタへの代入

(b) 任意のポインタによる間接参照を用いた書き込み

(c) 任意のグローバル変数への代入

(d) 任意の関数呼び出し

(e) asm関数の実行

*1 この処理ブロックとは、中括弧{...}で囲まれた次の(i)～(iii)のいずれかの文のうち、必ず記述順に実行される一連の文の集合を指す。
(i) 関数の文
(ii) if、else文で選択される文
(iii) while、do、for文で繰り返し実行される文

(5) 間接参照Bの後に、1つ以上の関数呼び出しを実行する。

(6) (5)の関数呼び出しの後に、(a)～(e)いずれかの文または式の中に間接参照Bと同一の間接参照Cが実行される。

(a) if～else文 で選択される文

(b) do～while文 で繰り返し実行される文

(c) while文の制御式

(d) for文の制御式

(e) 制御式が0以外の定数である while文、およびfor文で繰り返し実行される文

(7) 最初に間接参照B、次に関数呼び出し、最後に間接参照Cの順序で実行される。

(8) 間接参照Bが実行されてから、間接参照Cが実行される直前までに、次の(a)～(d)いずれの実行も存在しない。

(a) (2)のポインタへの代入

(b) 任意のポインタによる間接参照を用いた書き込み

(c) 任意のグローバル変数への代入

(d) while文 あるいは for文

2.2 発生例

[ソースファイル例1]
[sample1.c]

```
extern void extfunc1(void);
```

```
int func1(char *ptr, int cnt)  /* 条件(2) */
{
   int c1 = 0, c2 = 0;
   while (--cnt) {
                        /* 条件(4) */
      c1 = ptr[0];          /* 条件(3)(a) */
      extfunc1();           /* 条件(5) */
                        /* 条件(8) */
      if (cnt) {           /* 条件(6)(a),(7) */
                        /* 条件(8) */
         c2 = ptr[0];       /* 条件(6)(a),(7) */
      }
   }
   return c1 + c2;
}
```

[ソースファイル例2]

[sample2.c]

```
extern void extfunc2(void);
struct st2 { int a; int b; };

int func2(struct st2 *ptr, int cnt) /* 条件(2) */
{
   int c1 = 0, c2 = 0;
                        /* 条件(4) */
   c1 = ptr->b;             /* 条件(3)(b) */
   extfunc2();             /* 条件(5) */
                        /* 条件(8) */
   do {                   /* 条件(6)(b),(7) */
                        /* 条件(8) */
      c2 = ptr->b;          /* 条件(6)(b),(7) */
   } while (--cnt);
   return c1 + c2;
}
```

[ソースファイル例3]

[sample3.c]

```
extern void extfunc3(void);
```

```
int func3(char *ptr, int flg)  /* 条件(2) */
{
   int c1 = 0, c2 = 0;
   if (flg) {
                        /* 条件(4) */
      c1 = *(ptr+3);        /* 条件(3)(c) */
      extfunc3();           /* 条件(5) */
                        /* 条件(8) */
      while (*(ptr+3)) {    /* 条件(6)(c),(7) */
         ++c2;
      }
   }
   return c1 + c2;
}
```

[ソースファイル例4]

[sample4.c]

```
extern void extfunc4(void);

int func4(char *ptr, int cnt)  /* 条件(2) */
{
   int c1 = 0, c2 = 0;
                        /* 条件(4) */
   c1 = ptr[1];             /* 条件(3)(a) */
   extfunc4();              /* 条件(5) */
                        /* 条件(8) */
   for ( ; ptr[1]; --c2) {    /* 条件(6)(d),(7) */
   }
   return c1 + c2;
}
```

[ソースファイル例5]

[sample5.c]

```
extern void extfunc5(void);

int func5(char *ptr, int cnt)  /* 条件(2) */
{
   int c1 = 0, c2 = 0;
                        /* 条件(4) */
```

```
    c1 = *(ptr+0);          /* 条件(3)(c) */
    extfunc5();             /* 条件(5) */
                        /* 条件(8) */
    while (1) {             /* 条件(6)(e),(7) */
                        /* 条件(8) */
      c2 = *(ptr+0);        /* 条件(6)(e),(7) */
      if (--cnt) break;
    }
    return c1 + c2;
}
```

[オプション指定例]

```
 % cc32R -c -O4 sample1.c        /* 条件(1) */
 % cc32R -c -O4 sample2.c        /* 条件(1) */
 % cc32R -c -O4 sample3.c        /* 条件(1) */
 % cc32R -c -O4 sample4.c        /* 条件(1) */
 % cc32R -c -O4 sample5.c        /* 条件(1) */
```

(% は プロンプトを表します)

3. **回避策**

次のいずれかの方法で抑止できます。

(1) 処理ブロックの先頭でポインタ変数を作成し、そのポインタ変数による間接参照に変更する。

[ソースファイル例1の変更例]

[sample1.c]

```
extern void extfunc1(void);

int func1(char *dmy, int cnt)  /* 仮引数のポインタをdmyに変更 */
{
    int c1 = 0, c2 = 0;
    while (--cnt) {
      char *ptr = dmy;      /* ポインタptrを作成してdmyで初期化 */

      c1 = ptr[0];          /* 以降の間接参照は、
                            作成したポインタで行われる */
      extfunc1();

      if (cnt) {
```

```
        c2 = ptr[0];      /* 以降の間接参照は、
                          作成したポインタで行われる */
      }
    }
    return c1 + c2;
}
```

[ソースファイル例2の変更例]
[sample2.c]

```
extern void extfunc2(void);
struct st2 { int a; int b; };

int func2(struct st2 *dmy, int cnt) /* 仮引数のポインタをdmyに変更 */
{
    int c1 = 0, c2 = 0;
    struct st2 *ptr = dmy;    /* ポインタptrを作成してdmyで初期化 */

    c1 = ptr->b;              /* 以降の間接参照は、
                          作成したポインタで行われる */
    extfunc2();

    do {

       c2 = ptr->b;           /* 以降の間接参照は、
                          作成したポインタで行われる */
    } while (--cnt);
    return c1 + c2;
}
```

---

(2) 間接参照の直前にasm関数を記述する。
発生条件(3)の間接参照の直前に、asm関数を追加してください。(発生条件(4)が回避できます。)

[ソースファイル例3の変更例]
[sample3.c]

```
#pragma keyword asm on            /* asm関数を有効にする */
extern void extfunc3(void);
```

```
int func3(char *ptr, int flg)
{
   int c1 = 0, c2 = 0;
   if (flg) {
      asm("");                  /* asm関数を実行する */
      c1 = *(ptr+3);
      extfunc3();

      while (*(ptr+3)) {
         ++c2;
      }
   }
   return c1 + c2;
}
```

---

(3) -O4を含む最適化を抑止する。
-O4を含む最適化は、-O5,-O6,-O7,-Otimeのみ、または-Ospaceのみを指定しているときにも行われます。-Otimeまたは-Ospaceを使用する場合は、同時に-O0,-O1,-O2,-O3のいずれかを指定してください。

4. 恒久対策

本問題は、次期バージョンで改修する予定です。