お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

（注）本文中のURLは、以下の様に変更になりました。
http://tool-support.renesas.com/jpn/toolnews/shc/shcv7/dr_shcv71.html

2002年　11月　6日

― 日立半導体技術情報 ―

〒１００－０００４
東京都千代田区大手町2丁目6番2号
(日本ビル)
TEL　(03)5201-5022（ダイヤルイン）
株式会社 日立製作所 半導体グループ

製品分類	開発環境		発行番号	TN-CSX-044A	Rev.	第1版
題名	SuperH RISC engine C/C++コンパイラ Ver.7 不具合のご連絡(5)		情報分類	1．仕様変更 2．ドキュメント訂正追加等 ③．使用上の注意事項 4．マスク変更 5．ライン変更		
適用製品	P0700CAS7-MWR P0700CAS7-SLR P0700CAS7-H7R	対象ロット等 全ロット	関連資料	SuperH RISC engine C/C++コンパイラ、アセンブラ、最適化リンケージエディタ ユーザーズマニュアル ADJ-702-304A 第1版	有効期限 永年	

SuperH RISC engine C/C++コンパイラ Ver.7 に別紙に示す不具合があります。
次に示す製品を御使用のお客様につきましては周知願います。

型名	パッケージバージョン	コンパイラバージョン
P0700CAS7-MWR	7.0B	7.0B
	7.0.01	7.0.03
	7.0.02	7.0.04
	7.0.03	7.0.06
	7.1.00	7.1.00
P0700CAS7-SLR	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
	7.0.04	7.0.06
	7.1.00	7.1.00
P0700CAS7-H7R	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
	7.0.04	7.0.06
	7.1.00	7.1.00

なお、プログラムが本不具合に該当しているかを検出するチェックツールを以下よりダウンロードできます。

http://www.hitachisemiconductor.com/sic/jsp/japan/jpn/PRODUCTS/MPUMCU/TOOL/download/crosstool/release/rest_shcv7100.html

添付：P0700CAS7-021015

SuperH RISC engine C/C++コンパイラ Ver.7 不具合内容(5)

SuperH RISC engine C/C++コンパイラ Ver.7
不具合内容(5)

SuperH RISC engine C/C++コンパイラ Ver.7 台における不具合内容を以下に示します。
以下の不具合はチェックツールを使用することにより、プログラムに当該ケースが存在するか確認することができます。チェックツールは以下 URL より入手できます。

http://www.hitachisemiconductor.com/sic/jsp/japan/jpn/PRODUCTS/MPUMCU/TOOL/download/crosstool/release/rest_shcv7100.html

1. ループ内での不当ゼロ拡張生成

【現象】

ループ内に signed 型の変数 i から unsigned 型の変数 v の減算 i=i-v があるとき、不当にゼロ拡張を行う場合がある。

【例】

```
int i=319;
unsigned char v=97;
main()
{
    int f;
    for(f=0;f<5;f++){
        i = i - v;    /*  -v の結果がゼロ拡張されて演算される。*/
    }
}
```

【発生条件】

以下の条件をすべて満たす場合、発生することがあります。

(1) optimize=1 を指定している。
(2) ループ内に signed 型の変数 i から unsigned 型の変数 v の減算 i=i-v がある。
(3) 変数 i の型は 4 バイト(signed int, signed long)。
(4) 変数 v の型は 4 バイト未満の unsigned 型(unsigned char/short)。
(5) 変数 v はループ内不変式である。

【回避方法】

該当個所が存在した場合、以下のいずれかの方法で回避していただきますようお願いします。

(1) 該当ファイルを optimize=0 を指定してコンパイルする。
(2) 変数 i と v を同じ型にする。

2. 型変換不正

【現象】

char/short 型への型変換を行なった直後に浮動小数点型への型変換を行うと、char/short への変換が行われない場合がある。

【例】

[C ソース]

```
unsigned short US = 256;
int I;
float F;

main(){
    char c;

    c = US;        /* short 型変数を char 型変数に型変換 */
    I = 3 & c;     /* char 型変数がレジスタに割り付く */
    F = c;         /* char 型から float 型への型変換 */
}
```

[アセンブリソース]

```
_main:
    MOV.L       _US,R6
    MOV.L       _I,R5
    MOV.W       @R6,R0      /* 256 -> R0   char 型への型変換 EXTU が出力されない */
    LDS         R0,FPUL     /* 256 のまま float に変換している */
```

【発生条件】
以下の条件をすべて満たす場合、発生することがあります。
(1) CPU に SH2E または SH4 を指定している。
(2) char/short 型にそれより大きな型から型変換を行う。
(3) (2)の変換前の値が変換後の型の範囲を超えている。
(4) (2)の変換後の変数を浮動小数点型に型変換している。
(5) (2)の変換後の値がレジスタに割りついている。

【回避方法】
該当個所が存在した場合、以下の方法で回避していただきますようお願いします。
(1) 浮動小数点型への型変換する char/short 型変数を外部シンボルとし、レジスタに割り付かない様にする。

【注意】
チェックツールでプログラムに当該ケースが存在するか確認した場合、不具合に該当しない関数が検出される場合もあります。

3. FPSCR 設定コードをこえての命令不当移動
【現象】
cpu=sh4 を指定してコンパイルした場合に FPSCR の設定コードをこえて不当に FPU 命令をループ外に移動する場合がある。

【例】
[C ソース]

```
double dd;
struct tag {
    short aa ;
    long bb ;
    char cc:5 ;
} str ;
main()
{
    int i;
    str.bb = 10;
    str.aa = 10;
    for(i=5;i>=0;i--){
      str.cc =str.aa++   ;
      dd = str.bb ;
    }
}
```

[アセンブリソース]

```
_main:
        MOV.L       R14,@-R15
        MOV         #10,R5      ; H'0000000A
        MOV.L       L13+2,R2    ; _dd
        LDS         R5,FPUL
        MOV.L       L13+6,R7    ; _str
        FLOAT       FPUL,DR8                    ;  LDS R2 FPSCR をまたいでループ外に移動
        ADD         #8,R2
        MOV         #8,R1       ; H'00000008
                :
L11:
        MOV.B       @(8,R7),R0
        MOV         R0,R6
                :
        DT          R4
        OR          R1,R2
        LDS         R2,FPSCR
        ; FLOAT       FPUL,DR8                  移動前の位置
        ADD         #1,R5
                :
```

【発生条件】
以下の条件をすべて満たす場合、発生することがあります。
(1) optimize=1 を指定している。
(2) CPU に SH4 を指定している。
(3) fpu オプションを指定していない。

(4) ループ内に double 型の演算がある。
(5) (4)の演算がループ内不変式となる。

【回避方法】
該当個所が存在した場合、以下のいずれかの方法で回避していただきますようお願いします。
(1) 該当ファイルを optimize=0 を指定してコンパイルする。
(2) 該当ファイルを fpu=single/double を指定してコンパイルして、FPSCR の切替えコードが生成されないようにする。
(3) 対象となるループ内不変式をループ外に移動する。

【注意】
チェックツールでプログラムに当該ケースが存在するか確認した場合、不具合に該当しない関数が検出される場合もあります。

4. ループでのレジスタ割付不正
【現象】
最内側ループで割付けられたレジスタの内容が、そのループ内で破壊される場合がある。

【例】
複数の変数に対して同じレジスタを割り付けてしまい、レジスタの値が破壊されている。
[C ソース]

```
        :
    for(i1 = 0; i1 < max1 ; i1++) {
            a = b;
            for(i2 = 0; i2 < max2 ; i2++) {          /* (1) */
                    c = x + b;                       /* (2) */
            }                                        /* (3) */
            ans = a + c;
    }
        :
```

[アセンブリソース]

```
        :
     MOV.L   @(R0,R15),R5       ;a に R5 が割り付いている
        :
     MOV.L   R5,@(R0,R15)       ;a の退避
        :                       ;ループ入口(1)
     MOV.L   @(R0,R15),R5       ;c に R5 を割り付けた
        :
L1:
        :                        ;ループ本体(2)(a の参照無し)
     MOV     Rn,R5              ; c = x + b の結果を R5 に格納
        :
     BF   L1
        :                       ;ループ出口(3)
     MOV.L   @(R0,R15),R5       ;c を退避せずに R5 を上書き
        :
     MOV.L   R5,Rn              ;破壊された c の値をロードしている
        :
```

【発生条件】
以下の条件をすべて満たす場合、発生することがあります。
(1) optimize=1 を指定している。
(2) 最内側ループ内にレジスタ割り付け対象の変数がある。 (例の場合は変数 c)
(3) (2)の変数を含む式がループの外にありその同じ式にある変数(例の場合は変数 a)にも(2)の変数と同じレジスタが割り付く。(例の場合は R5 レジスタ)

【回避方法】
該当個所が存在した場合、以下のいずれかの方法で回避していただきますようお願いします。
(1)該当ファイルを optimize=0 を指定してコンパイルする。

5. FPSCR ロード不当削除
【現象】
double/float 切り替えコードが 2 つ出力されるとき、2 回目の FPSCR ロードが不正に削除される場合がある。

【例】
[C ソース]
```
double d0;
float f0, f1, f2;
main
{
    int i;
    for(i = 0; i < 100; i++){
      d0++;
      f1 = f1 + f0;
    }
}
```
[アセンブリソース]
```
_main:                                  ; function: main
    MOV.L       L13+2,R1    ; _d0
    MOVA        L13+6,R0
          :
L11:
    STS         FPSCR,R2
    DT          R6
    OR          R5,R2
    LDS         R2,FPSCR
    FADD        DR4,DR6             ; <--この後に STS FPSCR,R2 が必要
    AND         R4,R2
    LDS         R2,FPSCR
    BF/S        L11
    FADD        FR9,FR8
    ADD         #8,R1
          :
```

【発生条件】
以下の条件をすべて満たす場合、発生することがあります。
(1) optimize=1 を指定している。
(2) CPU に SH4 を指定している。
(3) fpu オプションを指定していない。
(4) double 及び float 型の演算がある。

【回避方法】
該当個所が存在した場合、以下のいずれかの方法で回避していただきますようお願いします。
(1) 該当ファイルを optimize=0 を指定してコンパイルする。
(2) 該当ファイルを fpu=single/double を指定してコンパイルして、FPSCR の切替えコードが生成されないようにする。

6. 関数コール引数の符号拡張不正
【現象】
戻り値を参照しない実行時ルーチンの引数に対して必要な符号拡張が行われない場合がある。

【例】
実行時ルーチン(__itod_a)への引数(unsigned long int) R0 に c を符号拡張した値を渡していない。
[C ソース]
```
double D;
int I;
unsigned short US = 256;
func2(){
    char c;
    c = US;
    I = 3 & c;
    D = c;
}
```
[アセンブリソース]
```
_func2:
            STS.L       PR,@-R15
            ADD         #-12,R15
            MOV.L       L20,R6      ; _US
```

```
MOV.L       L20+4,R5    ; _I
MOV.W       @R6,R2
MOV.L       L20+16,R6   ; _D
MOV         R2,R0
AND         #3,R0
MOV.L       R6,@R15
MOV.L       L20+20,R6   ; __itod_a
MOV.L       R0,@R5       ;<--この後に EXTS.B  R2,R2 が必要
JSR         @R6
MOV         R2,R0
```

【発生条件】
以下の条件を満たす場合、発生することがあります。
(1) 以下の実行時ルーチンを使用していて、その関数の引数は４バイトで、char/short の型の変数を引数に 渡している。
_itod_a、_utod_a

【回避策】
該当箇所が存在した場合、以下の方法で回避していただきますようお願いします。
(1)対象実行時ルーチンの引数に char/short を渡す前に、int の変数に入れてから引数として渡す。
【例】実行時ルーチン(__itod_a)への引数が char の場合

```
double D;
char C;
int I;
unsigned short US = 256;

func2(){
        char c;
        int temp;               /* 引数用の int 型変数の宣言 */

        c = US;
        I = 3 & c;
        temp = c;               /* char 変数を int 型の変数に代入する。*/
        D = temp;               /* int 型引数を__itod_a の引数とする。 */
}
```

【注意】
チェックツールでプログラムに当該ケースが存在するか確認した場合、該当個所が一箇所でも複数のメッセージが出力されることがあります。

7. 構造体メンバ設定/参照不正
【現象】
ポインタを介して２次元以上の構造体配列のある要素のメンバを参照をしたとき、そのメンバへ設定/参照が正しく行われない場合がある。

【例】

```
typedef struct {
    int aaa;
} ST;
void main()
{
    ((ST (*)[2])0x10000)[0]->aaa = 100;     /* 0x10000[0][0].aaa への設定が正しく行われない */
}
```

【発生条件】
以下の条件をすべて満たす場合、発生することがあります。
(1) 2 次元以上の構造体配列がある。
(2) ポインタを介して(1)の構造体配列のメンバを設定/参照している。

【回避策】
該当箇所が存在した場合、以下の方法で回避していただきますようお願いします。
(1) 構造体配列のアドレスを別のポインタ変数に代入して、ポインタの 1 次元配列としてメンバへの設定/参照を行う。

【例】

```
typedef struct {
    int aaa;
} ST;

void main()
{
    ST (*ptr)[2];                       /* ポインタ変数の宣言 */

    ptr = (ST (*)[2])0x10000;           /* 構造体配列のアドレスをポインタ変数に代入 */
    ptr[0]->aaa = 100;                  /* 代入した変数から aaa への設定を行う */
}
```