

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>

ルネサス 技術情報

〒 1 0 0 - 0 0 0 4
 東京都千代田区大手町 2 丁目 6 番 2 号
 (日本ビル)
 TEL (03)5201-5081 (ダイヤルイン)
 株式会社 ルネサスソリューションズ ツール技術部

製品分類	開発環境	発行番号	TN-CSX-056A	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(10)	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	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-702-444A 第 2 版	有効期限
	Ver.7.x 台	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
	7.1.01	7.1.01
	7.1.02	7.1.01
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
	7.1.01	7.1.01
	7.1.02	7.1.01
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
	7.1.01	7.1.01
	7.1.02	7.1.01
	7.1.03	7.1.02

なお、チェックツールを以下の URL より入手できます。

<http://www.renesas.com/jpn/products/mpumcu/tool/index.html>

添付 : P0700CAS7-031015J

SuperH RISC engine C/C++コンパイラ Ver.7 不具合内容(10)

SuperH RISC engine C/C++コンパイラ Ver.7 不具合内容(10)

SuperH RISC engine C/C++コンパイラ Ver.7 における不具合内容を以下に示します。

本不具合のチェックツールを以下 URL より入手できます。

<http://www.renesas.com/jpn/products/mpumcu/tool/index.html>

1. マクロ呼び出しでのインターナルエラー

【現象】

関数マクロ呼び出しを複数行にわたって記述し、その複数行の間に`//`コメントを行の 1 カラム目から記述した場合に、6001 インターナルエラーになります。

【例】

```
#define MACRO(a, b) a, b

void func() {
    MACRO(1,0
// <- 1 カラム目から`//`コメント
    );
}
```

【発生条件】

以下の条件をすべて満たした場合に発生します。

- (1) `listfile` オプションを指定している。
- (2) 関数マクロ呼び出しがある。
- (3) 関数マクロ呼び出し中に 1 カラム目から`//`コメントを記述している。

【回避方法】

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いただきますようお願いいたします。

- (1) `listfile` オプションを指定しない。
- (2) `//`コメントを 2 カラム目から記述する。
- (3) `//`コメントを`/* */`コメントにする。

2. C++限定付名称を`#pragma`に指定時のインターナルエラー

【現象】

C++言語の限定付名称を`#pragma`に指定した場合に、4099 インターナルエラーになる場合があります。

ただし、コンパイル環境(ホストコンピュータのオペレーティングシステム)によって再現しないことがあります。

【例】

```
class cPRINT {
public:
    static void IntPrint();
};
#pragma interrupt(cPRINT::IntPrint) /* 限定を含む名称の長さが 4 の倍数 */
void cPRINT::IntPrint(){}
```

【発生条件】

以下の条件をすべて満たした場合に発生することがあります。

- (1) C++言語を使用している。
- (2) `#pragma` 指定に限定付名称を指定している。
- (3) 限定を含む名称の長さが 4 の倍数である。

【回避方法】

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いたします。

- (1) 限定を含む名称の長さを4の倍数以外に変更する。

例：

```
cPRINT::InterruptPrint
```

3. align16 指定時の不当分岐(Ver.7.1.02 のみ)**【現象】**

align16 オプション指定時に、関数の最後が条件式で、関数呼び出しで終わっている節が複数あり、当該関数の後に別関数が続いている場合、分岐先不正になるかもしくはアセンブラ、最適化リンケージエディタでエラーになる場合があります。

【例】

```
#include <stdio.h>

void g(
    int a,
    int b,
    char *c);

void func(int x) {
    switch (x){
    case 0:
        g(0, 0, NULL);    /* 条件式の最後が関数呼び出し */
        break;
    default:
        g(0, 1, NULL);    /* 条件式の最後が関数呼び出し */
        break;
    }
}

main() {                /* 別関数が続く */
    func(0);
}
```

【発生条件】

以下の条件をすべて満たした場合に発生することがあります。

- (1) **optimize=1** オプションを指定している。
- (2) **align16** オプションを指定している。
- (3) 関数の最後が条件式であり、その条件式に複数の節がある。
- (4) (3)の節のすべてが関数呼び出しで終わっている。
- (5) 当該関数の後に、別関数が続いている。

【回避方法】

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いただきますようお願いいたします。

- (1) `optimize=0` オプションを指定する。
- (2) `align16` オプションを指定しない。
- (3) 最後の関数呼び出しの後に `nop0`組込み関数を追加する。

例：

```
#include <stdio.h>
#include <machine.h>          /* 追加 */

void func(int x) {
    switch (x){
    case 0:
        g(0, 0, NULL);
        break;
    default:
        g(0, 1, NULL);
        nop();                /* 追加 */
        break;
    }
}
```

4. DT 命令の不当生成

【現象】

`cpu=sh1` オプション以外指定時に比較を行った場合、DT 命令を不正に生成、または ADD 命令を不正に移動する場合があります。

【例 1】

```
struct tbl {
    char bit:1;
}S,*Sp=&S;
char a,b;
func(){
    int t;
    t = S.bit == 0;
    a = t << 1;
    if (t != 0 ) b++;
}

_func:
    MOV.L    L14,R6        ; _S
    MOV.B    @R6,R0
    TST      #128,R0
    MOVT     R6
                                ; (A)へ命令が不当に移動
    TST      R0,R0        ; -> TST R2,R2
    MOVT     R2
    MOV.L    L14+4,R5     ; _a
    SHLL     R2
    ADD      #-1,R6       ; (A) 不当に移動
    MOV.B    R2,@R5
    EXTS.B   R6,R2        ; (A) 不当に移動
    TST      R2,R2
    BF       L12
    MOV.L    L14+8,R6     ; _b
    MOV.B    @R6,R2
    ADD      #1,R2
    RTS
    MOV.B    R2,@R6
    :
```

【例 2】

```

char b;

func2(int c, char d) {
    c = d - 1;
    b = c << 1;
    if(c != 0) {
        b++;
    }
}

_func:
    MOV.L    L14+2,R6    ; _b
    EXTS.B   R5,R2
    SHLL    R4            ; d-1 ではなく引数の c をシフトしている
    DT      R2            ; c=d-1;および c!=0 の判定を行っている
    BT/S    L12
    MOV.B   R4,@R6
    ADD     #1,R4
    RTS
    MOV.B   R4,@R6
L12:
    RTS
    NOP

```

【発生条件】

以下の条件(1)、(2)を満たし、かつ(3a)または(3b)を満たした場合に発生することがあります。該当するかどうかはチェックツールを使用することにより確認することができます。

- (1) optimize=1 オプションを指定している。
- (2) cpu オプションに sh1 以外を指定している。
- (3a) ビットフィールドの 0 との'=='比較、もしくは'!='比較を行った後、その比較結果を再度比較している(例 1)。
- (3b) 加算結果を異なる型の変数に代入して(または異なる型にキャストして)比較を行っている(例 2)。

【回避方法】

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いただきますようお願いします。

- (1) optimize=0 オプションを指定する。
- (2) 発生条件(3a)の場合、ビットフィールド比較結果を、volatile 変数に代入した後、再度比較する。
- (3) 発生条件(3b)の場合、加算の結果を volatile 変数に代入する。

5. 左シフト時の不当拡張命令削除

【現象】

1 バイト、または 2 バイトデータを複合代入で左シフトした結果を参照した場合に、不当に符号拡張またはゼロ拡張命令を削除する場合があります。

【例】

```
short n;
void func(short s){
    sub(s <<= n);
}

_func:
    STS.L    PR,@-R15
    MOV.L    L11,R6          ; _n
    MOV.W    @R6,R1
    MOV.L    L11+4,R6       ; __sft1
    JSR     @R6
    EXTS.W   R4,R0
    MOV.L    L11+8,R2       ; _sub
    MOV     R0,R4          ; 演算結果が符号拡張されない
    JMP     @R2
    LDS.L    @R15+,PR
```

【発生条件】

以下の条件を全て満たした場合に発生することがあります。

該当するかどうかはチェックツールを使用することにより確認することができます。

- (1) `cpu=sh1|sh2|sh2e` オプションを指定している。
- (2) `unsigned/signed char/short` 型データを左シフトで複合代入している。
- (3) 複合代入した結果をシフトするデータの型よりも大きな型に代入している。
例えば、`int` 型の変数への代入や、パラメタとしての使用している場合。
- (4) 複合代入した結果が、そのシフト以降で使用されない。
- (5) シフト数に変数である。

【回避方法】

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いたします。

- (1) シフトの複合代入を止める。

6. 符号なし式の減算結果比較不正

【現象】

`unsigned` の型の変数や式どうしの減算結果を、`signed` の型にキャストして比較する場合に、不正に符号なし比較命令で比較を行う場合があります。

【例】

```
int test(unsigned int a, unsigned int b) {
    if ((int)(a - b) < 0) {
        return 1;
    } else {
        return 0;
    }
}

_test:
    CMP/HS  R5,R4          ; (unsigned int)a < (unsigned int)b で比較、
    MOVT   R0
    RTS
    XOR    #1,R0
```

【発生条件】

以下の条件をすべて満たした場合に発生することがあります。

該当するかどうかはチェックツールを使用することにより確認することができます。

- (1) `unsigned` の型の整数同士の減算の結果を、同じサイズの `signed` の型にキャストしてから比較している。
- (2) 減算結果をキャストした値が負である。

【回避方法】

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) 一度ローカル変数に代入してから比較するように修正する。

7. #pragma entry | noregsave 指定時のレジスタ退避・回復不正**【現象】**

`#pragma entry` もしくは `#pragma noregsave` を指定した場合、最適化リンケージエディタのレジスタ退避・回復最適化によりレジスタ退避・回復命令が不当に削除される場合があります。

【発生条件】

以下の条件をすべて満たした場合に発生することがあります。

- (1) `code=machinecode` オプションを指定している。
- (2) `goptimize` オプションを指定している。
- (3) `#pragma entry` もしくは `#pragma noregsave` を指定している関数内で、関数を呼び出している。
- (4) 最適化リンケージエディタで `optimize=register` オプションを指定している。

【回避方法】

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いただきますようお願いします。

- (1) `code=asmcode` オプションを指定する。
- (2) `goptimize` オプションを指定しない。
- (3) 最適化リンケージエディタで `optimize=register` オプションを指定しない。

8. 符号付きビットフィールドの範囲外定数値比較不正**【現象】**

符号付きビットフィールドの定数値比較(`==`または`!=`)において、定数値がビットフィールドの範囲を越えている場合に、比較結果が不正となる場合があります。

【例】

```
struct tbl {
    unsigned int ib1:1;
    int ib8:8;
} bf0 = {0,-2};

#include<stdio.h>
main() {
    if (bf0.ib8 != 254) {          /* 254 (0xFE) を -2 と判断してしまう */
        printf("OK¥n");
    }
}
```

【発生条件】

以下の条件(1)、(2)を満たし、かつ(3a)、(3b)、(3c)のいずれかを満たした場合に発生することがあります。
該当するかどうかはチェックツールを使用することにより確認することができます。

- (1) 符号付きビットフィールド(2～31bit)の定数値比較('=='または'!=')を行っている。
- (2) (1)の定数値が符号付きビットフィールドの範囲外であり、かつ最大値の2倍以下であり、かつ最小値の2倍より大きい。
例：8bitのビットフィールドの場合、-255～-129または128～254の範囲
- (3a) ビットフィールドのサイズが8bitでも16bitでもない。
- (3b) サイズが8bitのビットフィールドであり、そのビットオフセットが8、16、24ビットではない。
- (3c) サイズが16bitのビットフィールドであり、そのビットオフセットが16ビットではない。

【回避方法】

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いただきますようお願いします。

- (1) 比較対象となる定数値を符号付きビットフィールドの範囲内の値にする。
- (2) ビットフィールドを一度、int型の外部変数に代入し定数値と比較する。

以上