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

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-045A	Rev.	第 1 版
題名	H8S.H8/300 シリーズ C/C++コンパイラ 不具合のお知らせ		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	PS008CAS5-MWR PS008CAS4-MWR、 PS008CAS4-SLR、PS008CAS4-H7R PS008CAS3-MWR、 PS008CAS3-SLR、PS008CAS3-H7R	対象ロット等 全ロット	関連資料	H8S,H8/300 シリーズ 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-702-303 第 1 版 H8S,H8/300 シリーズ C/C++コンパイラユーザーズマニュアル ADJ-702-137D 第 5 版	有効期限 永年	

H8S,H8/300 シリーズ C/C++コンパイラ Ver. 1.0 から Ver.4.0.03 に不具合があります。詳細は添付資料を参照願います。

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。

H8S,H8/300 Series C/C++ コンパイラパッケージ (Windows 版) Ver.5.0, 5.0.01, 5.0.02

ホストコンピュータ (媒体)	H8S,H8/300 Series C/C++コンパイラパッケージ 型名
Windows (CD)	PS008CAS5-MWR

H8S,H8/300 Series C/C++ コンパイラパッケージ(Windows 版) Ver. 4.0, 4.0r1, 4.0A, 4.0Ar1, 4.0Ar2
H8S,H8/300 Series C/C++ コンパイラパッケージ(SPARC 版) Ver. 4.0, 4.0A, 4.0Ar1, 4.0B, 4.0.05, 4.0.06
H8S,H8/300 Series C/C++ コンパイラパッケージ(HP9000 版) Ver. 4.0, 4.0A, 4.0Ar1, 4.0B, 4.0Br1, 4.0.05, 4.0.06

ホストコンピュータ (媒体)	H8S,H8/300 Series C/C++コンパイラパッケージ 型名
Windows (CD)	PS008CAS4-MWR
SPARC Solaris (CD)	PS008CAS4-SLR
HP9000 (CD)	PS008CAS4-H7R

H8S,H8/300 Series C/C++コンパイラパッケージ(Windows 版) Ver. 3.0, 3.0A, 3.0B, 3.0Br1, 3.0C, 3.0Cr1
H8S,H8/300 Series C/C++ コンパイラパッケージ(SPARC 版、HP9000 版) Ver. 3.0, 3.0B, 3.0C

ホストコンピュータ (媒体)	H8S,H8/300 Series C/C++コンパイラパッケージ 型名
Windows (CD)	PS008CAS3-MWR
SPARC Solaris (CD)	PS008CAS3-SLR
HP9000 (CD)	PS008CAS3-H7R

記

お問い合わせ先: (株)日立製作所 半導体グループ マイコンテクニカルサポートグループ Email: micontech@sic.hitachi.co.jp
添付: (1) H8S,H8/300 シリーズ C/C++コンパイラ不具合のご報告　PS008CAS5-021007J　5枚

H8S,H8/300 シリーズ C/C++コンパイラ不具合のご報告

H8S,H8/300 シリーズ C/C++コンパイラに以下に示す不具合があります。これらは Ver. 4.0.04 で修正予定です。このバージョンのコンパイラは H8S,H8/300 シリーズ C/C++コンパイラパッケージ Windows 版 5.0.03 および UNIX 版 4.0.07 に同梱予定です。

１．単項マイナス演算子に関する不正コード

【対象バージョン】H8S,H8/300 シリーズコンパイラ Ver. 1.0 から Ver. 4.0.03
【不具合現象】
char/short/int 型の変数に対し，型変換（暗黙の型変換も含む）を行った後に単項マイナス演算を行った場合、
演算結果が不正となる場合があります。
【不具合例１】
[ソースプログラム]

```
char  c;
short s;
int   i;
long  l;
vois sub()
{
    s=-c;        /* c=-128   の時，s=-128    (正しくは s=128)   */
    i=-c;        /* c=-128   の時，i=-128    (正しくは i=128)   */
    l=-(long)c;  /* c=-128   の時，l=-128    (正しくは l=128)   */
    l=-(long)s;  /* s=-32768 の時，l=-32768  (正しくは l=32768) */
    l=-(long)i;  /* i=-32768 の時，l=-32768  (正しくは l=32768) */
}
```

[正しくない出力コード] 不具合例１の **s=-c;** の場合

```
MOV.B       @_c,ROL
NEG.B       ROL
EXTS.W      R0
MOV.W       R0,@_s
```

[正しい出力コード] 不具合例１の **s=-c;** の場合

```
MOV.B       @_c,ROL
EXTS.W      R0
NEG.W       R0
MOV.W       R0,@_s
```

【発生条件】
次の条件を全て満たす場合，発生いたします。
(1) 下記変数を次の型変換を行った後，単項マイナス演算を行った場合
char 型 → short,int,long へ型変換
short 型 → long へ型変換
int 型 → long へ型変換
(2) 上記変数が次の値の場合
char 型 → -128
short 型 → -32768
int 型 → -32768
【回避策】
次のいずれかの方法で回避願います。
(1) 型変換を実施してから，単項マイナス演算を実施します。
[修正例] 不具合例１の s=**-c;** の場合

```
s = c;
s= -s;
```

(2) 最適化を抑止(**-optimize=0** オプションを指定)します。

その他の不具合の例を以下に示します。
【不具合例２】

```
        char c;
        int  i1,i2;
        i1 = -c + i2;
```

【不具合例３】

```
        char c;
        int  func();
        int func()
        {
            return( -c );
        }
```

２．不正なＡＮＤ命令が生成される

【対象バージョン】H8S,H8/300 シリーズコンパイラ Ver. 1.0 から Ver. 4.0.03
【不良現象】
シフト演算後にビットごとＡＮＤ演算を行った場合，または連続して複合代入(&=,|=,^=)を行った場合，
コンパイル実行環境(【注】参照)によって不正なＡＮＤ命令が生成されることがあります。
【不具合例１】
[ソースプログラム]

```
  unsigned int X;
  sub( unsigned int Y )
  {
        X = (Y >> 14) & 0x2 ;
  }
```

[正しくない出力コード]

```
        MOV.W       @_Y,     R0
        ROTL.W      #H'2,    R0
        AND.L       #H'20002,ER0    <--- 正しくないコード
        MOV.W       R0,      @_X
```

[正しい出力コード]

```
        MOV.W       @_Y,     R0
        ROTL.W      #H'2,    R0
        AND.W       #H'2,    R0     <--- 正しいコード
        MOV.W       R0,      @_X
```

【不具合例２】

```
     int sub(int Y)
     {
        Y &= 0x3;
        Y &= 0x2;
        return Y;
     }
```

[正しくない出力コード]

```
        AND.L       #H'20002,ER0    <--- 正しくないコード
```

[正しい出力コード]

```
        AND.W       #H'2,    R0     <--- 正しいコード
```

【発生条件】
［パターン１］
次の条件をすべて満たす時，発生する場合があります。
(1) 変数に対し定数シフト演算を行った後，ビットごとＡＮＤ演算を行う。
(2) 変数とシフトの関係は下記の通り。
signed char/short/int の時、<< 演算
unsigned char/short/int の時、>> 演算
［パターン２］
次の条件をすべて満たす時，発生する場合があります。
(1) 変数と定数とのビットごとの複合代入演算(&=,|=,^=)を行う。
(2) 同一変数に対し、同種のビットごと定数複合代入演算を連続して記述する。
(3) 演算の型は[unsigned]char/short/int。
【注】コンパイラ解放済みメモリ領域の値に依存するため、同一プログラム同一オプションでもコンパイル環境

（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。
【回避策】
上記条件を満たすとき，次のいずれかの方法で回避願います。
［パターン１］
(1)シフト演算とビットごとＡＮＤ演算を２つの式に分割する。
そのとき，シフト演算の結果を一時 volatile 宣言された変数へ格納し、ＡＮＤ演算でその変数を使用する。
[修正例]

```
volatile unsigned int dummy;
dummy  = Y >> 14;
X = dummy & 0x2 ;
```

(2)最適化なし(-op=0)でコンパイルする。
［パターン２］
(1)連続した命令を１つにまとめて記述する。
[修正例]

```
Y &= 0x2;
```

(2)最適化なし(-op=0)でコンパイルする。

３．ループ内の不変式の括り出し最適化が効かない

【対象バージョン】H8S,H8/300 シリーズコンパイラ Ver. 3.0 から Ver. 4.0.03
【不良現象】
コンパイラの最適化の１つである“ループ内の不変式の括り出し最適化”が
コンパイル実行環境(ホストコンピュータのオペレーティングシステム)によって効かない場合があります。
【不具合例１】
[ソースプログラム]

```
for (  …  ;  …  ;  …  ){
        for (   …  ;   …   ;   …  )
              X =  0;
}
```

のようなループ式を書いた場合，最適化前の生成コードとして，下記のように出力します。
[最適化していない出力コード]

```
      <省略>
       BRA   L1            <----- (A)
 L2:
      <省略>
       BRA   L3
 L4:
      <省略>
       SUB.W       E0,E0    <----- (※)
       MOV.W       E0,@_X
 L3:
      <省略>
       Bcc   L4
 L1:
      <省略>
       Bcc   L2            <----- (B)
```

このとき，（※）で使用しているレジスタ E0 が（A）～（B）の範囲で
変更されていないのであれば，下記のように最適化（不変式の移動）を実施します。
[最適化した出力コード]

```
      <省略>
       SUB.W       E0,E0   <--------------
       BRA   L1            <----- (A)    |
 L2:                                     |
      <省略>                             | (移動)
       BRA   L3                          |
 L4:                                     |
      <省略>                             |
                           ---------------
       MOV.W       E0,@_X
```

```
L3:
      <省略>
       Bcc   L4
L1:
      <省略>
       Bcc   L2            <----- (B)
```

この最適化によって，ループ速度が向上します。
しかし，今回，この最適化が最適化条件を満たしているにもかかわらず効かない場合があります。

【発生条件】

次の条件をすべて満たす時，発生する場合があります。
(1) ループ文（for, while 文等）が存在する。
(2) ループ文の中に不変式（上記では　SUB.W E0,E0　が該当）が存在する。
(3) ループ文の中に，他の最適化によって削除されるコードが存在する。

【注】コンパイラ解放済みメモリ領域の値に依存するため、同一プログラム同一オプションでもコンパイル環境（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。

【回避策】

次の方法で回避願います。
(1)ソースを分割することにより，最適化が効くことがあります。

４．不正な分岐

【対象バージョン】H8S,H8/300 シリーズコンパイラ Ver. 1.0 から Ver. 4.0.03

【不良現象】

比較演算で，被比較式の演算結果がオーバーフローする場合、正常に比較分岐できない場合があります（ア）。
なお、型変換の記述により、コンパイラの最適化によって、演算の最適化（int 演算をすべきところ、char 演算を実施）が実施され、その結果オーバーフローになる場合も含まれます（イ）。

【不具合例１】（ア）の場合

```
int i1=1;
int i2=-32767;

if ( (i1 - i2) < 0)
    printf("OK\n");
else
    printf("NG\n");
```

※上記被比較式　**(i1 - i2)**　の結果は **-32768**　であり、“ OK ”にもかかわらず, “ NG ”となってしまいます。

[正しくない出力コード]

```
MOV.W       @_i1,R0
MOV.W       @_i2,R1
SUB.W       R0,R1
BGE         Ln
```

[正しい出力コード]

```
MOV.W       @_i1,R0
MOV.W       @_i2,R1
SUB.W       R0,R1
MOV.W       R0,R0           <--- 上の正しくないコードでは本命令が誤って削除されている
BGE         Ln
```

BGE 命令は CCR の V（オーバフロー）フラグの影響を受けます。この BGE 命令は上の MOV.W R0,R0 が CCR の V フラグを常にクリアすることを仮定して使われています。しかし、上の SUB R0,R1 は V フラグを変更することがあります。MOV.W R0,R0 を削除すると BGE 命令を実行する時に V フラグが常にクリアされているとは限りません。

【不具合例２】（イ）の場合

```
char c1=1;
char c2=-127;

if ( (char)(c1 - c2) < 0)
    printf("OK\n");
else
    printf("NG\n");
```

※上記被比較式　**(char)(c1 - c2)**　の結果は **-128**　であり, “ OK ”にもかかわらず, “ NG ”となってしまいます。
この場合，**(c1-c2)**は ANSI 仕様によると int 型演算を実施するので，オーバーフローはしませんが、

コンパイラの最適化で小さいサイズ(char 型)で演算されることにより、オーバーフローの演算となります。

【発生条件】

次の条件をすべて満たす時，発生すします。

（1）定数との比較演算の式を記述する。

```
if ( （被比較式）op 定数 )
```

（2）（1）の op と　定数の関係は下記である。

op	定数
<	0 or 1
>	0 or 1
<=	0
>=	0

（3）被比較式の記述は下記のいずれかである。

```
(char) ([unsigned]char {+|-} [unsigned]char) /* 一方が定数の 1or2 の場合も含む */
       (int  {+|-} int)                      /* 一方が定数の 1or2 の場合も含む */
(int ) ([unsigned]int  {+|-} [unsigned]int)  /* 一方が定数の 1or2 の場合も含む */
(long )([unsigned]int  {+|-} [unsigned]int)  /* 一方が定数の 1or2 の場合も含む */
       ( long  {+|-} long)                   /* 一方が定数の 1or2 の場合も含む */

(char) (-char)  /* 値は -128 */
       (-int)   /* 値は -32768 */
       (-long)  /* 値は 0x80000000 */

       (char++) /* 値は 127 */
       (int++)  /* 値は 32767 */
       (long++) /* 値は 0x7fffffff */

       (char--) /* 値は -128 */
       (int--)  /* 値は -32768 */
       (long--) /* 値は 0x80000000 */
```

【回避策】

被比較式を比較文の前に実施するよう変更します。

[修正例] （ア）の場合

```
int i1=1;
int i2=-32767;
int i3;

i3 = i1 - i2;
if ( i3 < 0)
    printf("OK\n");
else
    printf("NG\n");
```

[修正例] （イ）の場合

```
char c1=1;
char c2=-127;
char c3;

c3=(char)(c1 - c2);
if ( c3 < 0)
    printf("OK\n");
else
    printf("NG\n");
```

以上