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

2003 年　11 月　27 日

― ルネサス 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)5201-5081 (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-059A	Rev.	第１版
題名	H8S,H8/300 Series C/C++コンパイラパッケージ リビジョンアップのお知らせ		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	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-702-303A 第 1 版	有効期限 永年	

H8S,H8/300 Series C/C++ コンパイラパッケージを Ver.5.0.06 (PS008CAS5-MWR)、
Ver.4.0.05 (PS008CAS4-MWR)、Ver.4.0.09 (PS008CAS4-SLR、PS008CAS4-H7R)にリビジョンアップいたしました。
詳しい修正内容については、添付資料の PS008CAS5-031113J をご参照ください。

WindowsR版のお持ちの御客様は、アップデータを以下の URL より入手できます。

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

UNIX 版のお持ちの御客様はリビジョンアップ依頼を販売元までご連絡下さい。

添付：PS008CAS5-031113J
H8S,H8/300 Series C/C++ コンパイラパッケージ **アップデート内容**

H8S,H8/300 Series C/C++コンパイラパッケージ
アップデート内容

本リビジョンにおけるアップデート内容(不具合修正)を下記に示します。

1. High-performance Embedded Workshop (PS008CAS5-MWR のみの変更)

1. 1 エミュレータ用プロジェクトの表示

日本語版の HEW をインストールしたときに、エミュレータ用プロジェクト(プロジェクト名：Debugger only -****)が表示するように改善しました。

1. 2 ネットワークデータベース注意事項の対策

Windows(R) ME で表示されるネットワークデータベースエラーメッセージボックスの表示を抑止しました。

1. 3 プロジェクトジェネレータ生成データの追加、修正

新たに以下の CPU のプロジェクト生成を追加しました。

H8XS/1650、H8/36014F、H8/36024F、H8/36037F、H8/36057F、H8/3694F、
H8/38000、H8/38001、H8/38002F、H8/38004F

また、以下の CPU の I/O 定義ファイル(iodefine.h)を修正しました。

H8/3687

2. コンパイラ

[機能改善]

1) **switch 数の制限値拡張**
1 ファイル内の switch 数の上限を 256 から 2048 に変更しました。

[不具合修正]

1) **union 初期値不正**
char 配列をメンバに持つ共用体型 auto 変数の初期値に、配列サイズを超える長さの文字列リテラルを{ } なしで指定すると、エラーとならずに文字列全体をコピーするコードが生成され、スタックを破壊してしまう問題点を修正しました。

[例]

```
void test()
{
    union {char c[4];} x = "long string";        /* 配列に収まりきらないが、エラーとならない */
}
```

[発生条件]
次の条件をすべて満たす時、発生します。
ア) char / unsigned char 型配列をメンバに持つ共用体型 auto 変数が存在する。
イ) ア)の変数の初期値に、配列サイズを超える長さの文字列リテラルを{ }なしで指定している。

2) 無名ビットフィールド不正
構造体の先頭に無名ビットフィールドがあり、その直後のメンバが配列または構造体である構造体に初期値を設定した場合、無名ビットフィールドのギャップ分が出力されない問題を修正しました。

[例]
```
struct S {
   char :1;
   char a[3];
} s = {"abc"};
```

[出力コード]
・正しいオブジェクト
```
   _s:
         .data.l h'00616263
```

・不正なオブジェクト
```
   _s:
         .data.l h'61626300            ; 無名ビットフィールドのギャップが出力されない
```

[発生条件]
次の条件をすべて満たす時、発生します。
ア) 構造体の先頭メンバに無名ビットフィールドを宣言している。
イ) ア)の直後のメンバが、配列または構造体を宣言している。
ウ) 上記構造体変数に初期値を設定している。

3) ポインタ型 cast 不正
H8/300 またはノーマルモード時の定数式演算において、定数値をポインタ型へキャスト時、定数値の上位ワードをクリアせず、その後の演算で結果が不正となる場合がある問題点を修正しました。

[例]
```
   long x = (long)(char *)0x12345678;
```

[出力コード]
```
   .section D,data
_x:
   .data.l h'12345678      /* 上位 2byte がクリアされません */
```

[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。
ア) cpu=300,300l,300hn,2000n または 2600n を指定している。
イ) 定数値をポインタ型へキャストしている。
ウ) イ)の定数値が 2byte を超えている。

4) 分岐削除不正
switch 文を記述した場合、不正に分岐する場合がある問題点を修正しました。

[例]

```
typedef struct { char a; char* b; char* c; }st;
void func(st x)
{
   int i;
   switch(x.a)
   {
   case 0:
         func1(x->b);       /* 共通式 */
         func1(x->c);
            :
         break;
```

```
  case 1:
       func1(x->b);       /* 共通式 */
         :
       break;
  case 2:
       func1(x->b);       /* 共通式 */
       break;
  case 3:
       func1(x->b);       /* 共通式 */
       break;
       <以下省略>
```

[不正コード]

```
  switch(x.a)
       MOV.W     @ER6,R1
       CMP.W     #-3879,R1
       BEQ       L1647:16
            :
       CMP.W     #-3884,R1
       BEQ       L1671:16        <---- (本来 L1673 に分岐)
       CMP.W     #-3883,R1
       BEQ       L1671:16
            :
            :
  L1671:
            :
  L1673:
```

[発生条件]

switch 文に case 文が複数あり、各 case 文内の式に共通式が存在する場合、発生することがあります。

5) ビットフィールド判定でオブジェクト不正

1 ビットのビットフィールドの値を判定する式があり、かつその前後で 0 を設定する式が記述された場合、0 の設定コードが削除され、オブジェクト不正となる場合がある問題を修正しました。

[例]

```
unsigned char X,Y;
struct{
     unsigned char F1:1;
     unsigned char F0:1;
}BIT1,BIT2;
void test(void){
     X = 0;
     if( BIT1.F0 ){                          /* 1 ビットのビットフィールドの値を判定します          */
          if( BIT2.F0 != (Y==0 ? 0 : 1) ){   /* 3 項演算子内で 0 を代入する式が記述されているため  */
               BIT2.F0 = (Y==0 ? 0 : 1);     /* オブジェクト不正となる可能性があります。            */
          }
     }
}
```

[不正コード]

```
BIT2.F0 = (Y==0 ? 0 : 1);
                                             ; sub.b     r0l,r0l の命令を不正に削除
  mov.b    r1l,r0h
  beq      l12:8
  mov.b    #1,r0l
l12:
  bld.b    #0,r0l
  bst.b    #6,@_bit2:32
```

[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。
ア) 1 ビットフィールドの判定式が存在する。
イ) ア)の式の前後に 0 を設定する式が存在する。

6) unsigned long 比較でオブジェクト不正
unsigned long 型への型変換を実施した変数と、unsigned long 型の定数を比較（<、<=、>、>= ）した場合、結果が不正となる場合がある問題点を修正しました。

[例]
```
unsigned int   ui1;
int sub()
{
  if (((unsigned long)ui1) >   0x80000000L )
       return(0);
  return(1);
}
```

[不正コード]
```
_sub:
  sub.w    r0,r0                    ; 判定式が常に真となり、リターン値が不正となります。
  rts
```

[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。
ア) (unsigned long)変数と定数の比較（<、<=、>、>= ）を行っている。
イ) ア)の変数が unsigned char、unsigned short、unsigned int のいずれかである。
ウ) 定数が 0x80000000 ～ 0xFFFFFFFF の範囲内のである（ -2147483648 ～ -1)。
（※但し、0xFFFFFFFF(-1)の場合の <=-1、 >-1 は除く）

7) ビットフィールド演算不正
同一データ領域内のビットフィールドへの設定が連続して記述された場合、不正なオブジェクトが生成される場合がある問題点を修正しました。

[例]
```
struct{
  int s1 :1;
  int s2 :3;
  int s3 :1;
  int s4 :3;
}bit;
void func()
{
  if (x)
       bit.s1=1;
  else
       bit.s2=2;
}
```

[出力コード]

<不正コード>
```
_func:
  mov.w r0,r0
  beq    l5:8
  mov.l  #_bit,er0
  mov.w @er0,r1
  and.w  #-20481,r1
```

<正常コード>
```
_func:
  mov.w    r0,r0
  beq      l5:8
  bset.b   #7,@_bit:32        ; ビットが不正に統合されます。
```

```
   bra     l7:8                        rts
l5:                                 l5:
   mov.l   #_bit,er0                   mov.b    @_bit:32,r0l
   mov.w   @er0,r1                     and.b    #-113,r0l
   or.w    #-24576,r1                  or.b     #32,r0l
l7:
   mov.w   r1,@er0                     mov.b    r0l,@_bit:32
   rts     rts
```

[発生条件]
同一データ領域内のビットフィールドへの設定を以下の条件の(A)、(B)の形で記述した場合、発生する場合があります。
ア) if 文

```
if(- - -)
      (A)
else
      (B)
```

イ) while 文

```
while((A))
      (B)
```

ウ) カンマ式

```
((A) ? (B) : EXP)
(EXP ? (A) : (B))
```

8) ビットフィールド設定不正

long 型のビットフィールド（ビットオフセット+ビットサイズ<=16 またはビットオフセット>16）へ定数値を設定する時、正しく設定できない場合がある問題点を修正しました。

＜例＞

```
#include <stdio.h>

struct ST {
     char C1;
     char C2;
     long UL1:6 ;
     long L1 :4 ;
}st;
void main(){
     st.C1 = 10;
     st.C2 = 10;
     st.L1 = 7 ;
     printf("st.L1 : %ld ¥n",st.L1);
}
```

[出力コード]

```
          .cpu      2600a:24
          .section  p,code,align=2
_main:
          push.l    er6
            :
          /* st.L1 = 7 ;    */
          /* 不正コード */                    /* 正しいコード */
          and.b     #-4,r0h         mov.w     @(2:16,er6),e0     ; メンバの参照・設定コードが
          and.b     #63,r0l         and.w     #-961,e0           ; 不正に削除されます。
          or.b      #1,r0h          or.w      #448,e0
          or.b      #-64,r0l        mov.w     e0,@(2:16,er6)
            :
          pop.l     er6
          rts
```

[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。

ア) long / unsigned long 型のビットフィールドへの設定コードで、ビットオフセット+ビットサイズ<=16、またはビットオフセット>16 である。

イ) 設定値が定数である。

ウ) 最適化有り(-optimize=1、デフォルト)指定している。

エ) 300、300L 以外の CPU 指定している。

9) ビットフィールドの乗算の複合代入

代入先がビットフィールドの乗算の複合代入演算式を記述した場合、生成コードが不正となる場合がある問題点を修正しました。

[例]

```
struct ST {
   unsigned int bit1 :6;
   signed int bit2 :3;
} st1,st2,*stp;

sub()
{
   //省略
   stp->bit1 *=   st2.bit2;
```

```
   //省略
}
```

[不正コード]

```
   mov.w    @_st2:32,r0
   mov.w    #1539,r1
   jsr      @$bfsi$3:24          // st2.bit2 の値を r0 に設定
   mov.l    @_stp:32,er2
   mov.b    @er2,r1l
   shlr.b   #2,r1l
   extu.w   r1                   // stp->bit1 の値を r1 に設定
   mov.w    r0,@(2:16,sp)        // r0 を退避
   mov.w    r1,r0                // r0 を破壊
   mulxu.w  r0,er0               // 破壊された r0 を st2.bit2 として使用
        :
```

[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。
ア) 乗算の複合代入演算式が存在している。
イ) ア)の代入先の変数がビットフィールドであり、そのビットフィールドがポインタ参照である。
ウ) -regexpansion(デフォルト)を指定している。
エ) 最適化(-optimize=1、デフォルト)を指定している。
オ) レジスタ変数が[E]R3～[E]R6 全て使用されている。

10) カンマ式での変数への設定コード未出力

カンマ式の第 1 子に、その結果を使用しないビットごとの論理演算の(&, ^, |)を記述した時、ビット演算の式の中に変数への設定値があった場合でも、その設定値が削除されてしまう問題点を修正しました。

[例]

```
int   i1,i2,i3;
sub()
{
    ((long)(i2++) & (i3=1)) , i1 = 8; /* i2++, i3=1 が不正に実行されない */
}
```

[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。
ア) カンマ式の第 1 子に、その結果を使用しないビットごとの論理演算を記述している。
イ) ア)の第１子、および第２子に変数を更新する式の記述がある(一方が構造体のメンバ記述でも該当)。
ウ) ア)の第１子、または第２子に型変換(cast)の記述がある。

11) ローカル変数のデバッグ情報不正

無限ループのある関数で、スタック上に割付られた局所変数や引数のデバッグ情報が誤ってしまい、正しくシンボルデバッグができない問題点を修正しました。

[例]

```
extern int a;
void func()
{
   char c[4];                    /* c の変数のシンボルデバッグができない          */
   c[3] = 1;

   for(;;){
        a++;
   }
}
```

[発生条件]

次の条件をすべて満たす時、発生します。

ア) debug オプションと optimize オプションを指定している。

イ) 関数内に無限ループが存在し、局所変数または引数がスタック上に割りついている。

以上