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　1 月　31 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-048A	Rev.	第１版
題名	SuperH RISC engine C/C++コンパイラ Ver.7.1.01 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	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++コンパイラパッケージ Ver.7.1.01 にリビジョンアップしました。
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

添付：P0700CAS7-030114J
SuperH RISC engine C/C++ Compiler Package
Ver.7.1.01 アップデート内容

SuperH RISC engine C/C++ Compiler Package
Ver.7.1.01 アップデート内容

本パッケージのアップデート内容(不具合修正および機能追加)を以下に示します。

ただし、項番 1 は PC 版のみです。

1. Hitachi Embedded Workshop (PC 版)

1.1 メニュー、ダイアログの日本語化対応

Hew のメニューやダイアログが日本語表示になりました。

1.2 ドラッグ&ドロップのサポート

Watch ウィンドウへの変数の登録をエディタウィンドウからマウス操作によるドラッグ&ドロップで可能になりました。

1.3 Hew サーバの公開

COM 技術による Hew サーバを公開しました。 本サーバは Out-of-process サーバとして機能します。

1.4 キャッツ(株)製 ZIPC ツールとの接続

キャッツ(株)製 ZIPC ツールと連動してプログラムのデバッグが可能になりました。

1.5 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。

SH7705

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

SH7046, SH7727

2. コンパイラ

2.1 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不具合を解決しました。

＜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。

2.2 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なうと、char/short への変換が行われない不具合を解決しました。

＜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。

2.3 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不具合を解決しました。

＜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。

2.4 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不具合を解決しました。

＜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、レジスタ内容が破壊されている

[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(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)

2.5 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不具合を解決しました。

＜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。

2.6 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不具合を解決しました。

＜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】

以下の条件をみたす場合、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

2.7 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したとき、そのメンバへ設定/参照が正しく行われない不具合を解決しました。

＜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。

2.8 latin1指定時不当なエラー出力

【現象】

latin1 を指定してコンパイルした際に、一部正常な latin1 コードに対して不当にエラーメッセージを出力する不具合を修正しました。

2.9 スタックサイズ表示不正

【現象】

割り込み関数のスタックサイズ情報(リスティングファイル/アセンブリソースのスタックフレームサイズ)に、CPU が暗黙的に使用する PC と SR の退避分が含まれていない不具合を解決しました。

【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 SH1/SH2/SH2E が指定されている。

(2) #pragma interrupt 指定で指定した割込み関数がある。

2.10 volatile 変数へのアクセス

volatile 変数アクセスに関する以下の制限を改善しました。

(a) 連続した volatile 変数アクセス

【ソース】

```
void main(void) {
      *((volatile unsigned char*)0xfffff000);       /* (1) */
     *((volatile unsigned short*)0xfffff004);      /* (2) */
     *((volatile unsigned short*)0xfffff004) = 1;   /* (3) */
      /*   (1)、(2)、(3)の順序で参照することを保証   */
}
```

(b) volatile ポインタへのキャストを介した構造体メンバ参照

【ソース】

```
struct st_tmu2 {
      unsigned int TCOR;
     unsigned int TCNT;
};

#define TMU2 (*(volatile struct st_tmu2 *)0xFFD80020)

void time_wait (unsigned long time) {
    volatile unsigned long tcnt;
    unsigned long tcnt_copy, time_cnt;

    time_cnt = time;
     tcnt_copy = tcnt = TMU2.TCNT;
     while ((tcnt_copy - tcnt) < time_cnt) {
          tcnt = TMU2.TCNT;
          /* ループの中で毎回代入を行うことを保証 */
     }
}
```

2.11 _sectop,_secend 指定時セクション属性の変更

【現象】

__sectop、__secend 指定時セクション名の頭文字が 'P' の時は属性を CODE とするように変更し、リンク時に L1323 が出力されないようにしました。

3. 最適化リンケージエディタ

3.1 内部エラーの解決

以下の内部エラーが生じる不具合を解決しました。

- 共通コード統合最適化指定時の内部エラー(1703)
- 定数/文字列統合最適化指定時の内部エラー (1704)
- 分岐命令最適化指定時の内部エラー (8899)

3.2 form={binary | stype | hexadecimal}指定時の不正動作

output オプションで存在しないディレクトリを指定した際、binary/stype/hexadecimal 形式で出力指定をした場合に、出力ファイルが作成されないにも関わらずエラーメッセージが出力されない問題を解決しました。

3.3 未参照シンボル削除最適化指定時のオブジェクト不正

下記条件を満たす場合、最適化によってアクセスする配列の要素が不正となる場合がある不具合を解決しました。

(1) アクセスされる配列 A_arr[]が存在する。
(2) 最適化にて削除対象となる配列 B_arr[](もしくは変数)が存在する。
(3) A_arr[],B_arr[]はそれぞれ別セクションに存在している。
ここではそれぞれのセクションを A,B とする。
(4) start オプション指定により、A セクションと B セクションのアドレスが重複するようにセクションを配置する。
(5) 未参照シンボル削除最適化を有効にする。

3.4 C++オブジェクトをリンクする際の不正なエラー出力

C++で、template を使用して作成したオブジェクトをリンクした際に、不正に P3300(F)エラーが生じる場合がある不具合を解決しました。

3.5 SYSROF 形式へ変換した際のデータ欠如

下記条件を満たす場合、データが不正に欠落してしまう場合がある不具合を解決しました。

(1) C++によりソースが記述されている。
(2) リンク時に、未参照シンボル削除最適化を有効にしている。
(3) コンバータによりオブジェクトフォーマットを ELF->sysrof に変換している。

以上