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　9月　9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-043A	Rev.	第1版
題名	SuperH RISC engine C/C++コンパイラ Ver.7.1.00 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	対象ロット等	関連資料	有効期限
P0700CAS7-MWR P0700CAS7-SLR P0700CAS7-H7R	全ロット	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	永年

SuperH RISC engine C/C++コンパイラパッケージ Ver.7.1.00 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
P0700CAS7-SLR	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
	7.0.04	7.0.06
P0700CAS7-H7R	7.0B	7.0B
	7.0.02	7.0.03
	7.0.03	7.0.04
	7.0.04	7.0.06

添付：P0700CAS7-020826J
SuperH RISC engine C/C++ Compiler Package
Ver.7.1.00 アップデート内容

SuperH RISC engine C/C++ Compiler Package
Ver.7.1.00 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 1,2 は PC 版のみ、項番 7,8 は UNIX 版のみです。

1. Hitachi Embedded Workshop (PC 版)

1.1 プロジェクトのロードとアンロード

ワークスペース内で複数のプロジェクトを利用している場合、ワークスペースオープン時はカレントプロジェクトのみロードされ参照可能になります。他のプロジェクトを参照する場合は、ワークスペースウィンドウ上で該当するプロジェクトを選択し、マウス右ボタンでポップアップメニューの[Load Project]を指定することで参照可能になります。

また、参照不要となったプロジェクトは、同様の操作でポップアップメニュー[Unload Project]を指定することで、HEW 上の管理から一時的に開放されます(メモリの節約になります)。

1.2 ワークスペースウィンドウ Navigation タブ設定

ワークスペースウィンドウの Navigation タブ上でポップアップメニュー(マウス右ボタンをクリック)の[Configure view...]を選択すると、Navigation タブ上で表示する情報を設定することができます。初期設定では、"ANSI C Functions"と"C Defines"を表示します。

1.3 印刷時のヘッダ、フッダ設定

[File - Page Setup...]メニューで、印刷時のヘッダ、フッダの設定が可能になりました。

1.4 カスタムプレースフォルダの設定

[Tools - Customize]メニューの Placeholder タブで、ユーザ固有のプレースホルダの設定が可能になりました。HEW 全体で使用する場合は、"Application wide custom placeholders:"に、またワークスペース個別に使用する場合は、"Workspace wide custom placeholders:"に設定してください。

1.5 MAP 最適化対応の自動ビルド

HEW2.0 では、C コンパイラで最適化リンケージエディタが出力した外部シンボル割り付け情報を活用した最適化を実行するためにカスタムフェーズを提供していましたが、HEW2.1 では、オプション設定だけで最適化ビルドが実行可能になりました。

ツールチェインオプションダイアログの C/C++タブで Optimize として"Include map file"を指定すると、最適化リンケージエディタ実行後、自動的に再度 C コンパイラからのビルド処理を実行します。

1.6 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。

SH7290

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

SH7727

1.7 スタック解析ツールにデータマージ機能追加

ユーティリティツールの一つ、スタック解析ツールにデータマージ機能を追加しました。これにより、既に作成済のアセンブリプログラム等のスタックデータと最適化リンケージエディタ　　が生成したスタック情報ファイルのデータを合成できるようになりました。

1.8 プロジェクトジェネレータ生成データの修正

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

SH7622 : st_scif0 定義

2. SuperH RISC engine シミュレータ・デバッガ (PC 版)

2.1 セッションサポート (Ver. 8.0.01 -> Ver. 8.1.00)

ターゲット固有の情報はセッションに保存します。1 つのコンフィギュレーションに複数のセッションを持つことが可能です。

2.2 コマンドバッチファイル実行順序の指定 (Ver. 8.0.01 -> Ver. 8.1.00)

コマンドバッチファイルを自動実行する順番を指定できます。

実行するタイミングは、ターゲット接続時、ユーザプログラムロード前、後から選択できます。

2.3 Source ウィンドウの強化 (Ver. 8.0.01 -> Ver. 8.1.00)

Source ウィンドウ上に対応するアドレスとカバレッジ情報を表示します。また、表示／非表示も選択できます。

C ソースレベルでカバレッジ情報を表示することが可能になります。

2.4 カバレッジ機能強化 (Ver. 8.0.01 -> Ver. 8.1.00)

カバレッジ情報のセーブ／ロードをサポートします。過去のカバレッジ情報とのマージが可能になります。

2.5 Watch 機能の強化 (Ver. 8.0.01 -> Ver. 8.1.00)

Watch 情報のリアルタイム表示をサポートします。プログラム実行中にリアルタイムに情報

を　　更新します。また、Watch 情報のファイルへのセーブをサポートします。

2.6 Trace 機能の強化 (Ver. 8.0.01 -> Ver. 8.1.00)

Trace 取得情報の統計解析機能をサポートします。

解析する項目を指定することにより、どの情報が何回取得されたかを表示することが可能になります。

2.7 Trigger ウィンドウサポート (Ver. 8.0.01 -> Ver. 8.1.00)

Trigger ウィンドウをサポートしました。これにより、任意のタイミングで擬似割り込みを発生することが可能になります。

2.8 SH3-DSP(core)シミュレータサポート (Ver. 8.0.01 -> Ver. 8.1.00)

SH3-DSP(core)シミュレータをサポートしました。ASIC 向け SH3-DSP のデバッグが可能になります。

3. コンパイラ

3.1 オプションの追加 (Ver.7.0.04 -> Ver.7.0.06)

以下のオプションを追加しました。

(1) global_volatile={0 | 1}
　　外部変数の volatile 化

(2) opt_range={all | noloop | noblock}
　　外部変数最適化範囲指定

(3) del_vacant_loop={0 | 1}
　　空ループ削除

(4) max_unroll=<数値>
　　ループ最大展開数の指定

(5) infinite_loop={0 | 1}
　　無限ループ前の式削除

(6) global_alloc={0 | 1}
　　外部変数のレジスタ割り付け

(7) struct_alloc={0 | 1}
　　構造体/共用体メンバのレジスタ割り付け

(8) const_var_propagate={0 | 1}
　　const 定数伝播

(9) const_load={inline | literal}
　　定数ロードの命令展開

(10) schedule={0 | 1}

命令並べ替え

3.2 R0 レジスタの不正破壊 (Ver.7.0.04 -> Ver.7.0.06)

スタック渡しのパラメタがあるとき、R0 を不正に書き換える場合がある不具合を対策しました。

＜例＞

```
short func1(short a0, int *a1, int a2, short a3, short a4, short a5, short a6, int a7, int a8, int a9);
void func0(short a0, int *a1, int a2, short a3, short a4, short a5, short a6) {
                        :
    r1=func1(0,a1,0,0,0,0,0,0,0,0);
    if((r1>0)&&(r1!=1)) {
        func1(a0,a1,0,a3,a4,a5,a6,0,0,0);
    }
                        :
}

                        :
    MOV.L       R0,@(32,R15)        ; -> R0 を@(32,R15)に退避
    MOV         R8,R5
    MOV         #66,R0              ; -> R0 を破壊
    MOV.W       @(R0,R15),R3
    MOV         R9,R6
    MOV         #70,R0              ; -> R0 を破壊
    MOV.W       @(R0,R15),R1
    MOV         R0,R4               ; ->MOV.L @(32,R15),R4 を MOV R0,R4 に置換えコード不正
    MOV.L       R3,@(4,R15)
    MOV.L       R1,@(8,R15)
    MOV.L       R9,@(12,R15)
    MOV.L       R9,@(16,R15)
    BSR         _func1
    MOV.L       R9,@(20,R15)
                        :
```

【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) 当該関数にスタック渡しのパラメタが存在する。

3.3 BRA 命令飛び先不正 (Ver.7.0.04 -> Ver.7.0.06)

無条件分岐を含むプログラムにおいて、分岐を BRA 命令で行いかつ飛び先までの距離が 4094 バイトの時、飛び先が不正になるコードを生成する場合がある不具合を対策しました。

【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) code=machinecode を指定している。または code オプションを指定していない。

(2) BRA 命令から飛び先までの距離が 4094 バイトである。

3.4 PR レジスタ退避・回復命令の不当削除 (Ver.7.0.04 -> Ver.7.0.06)

以下 C ソースを speed オプションでコンパイル時、不当に PR レジスタの退避・回復命令を削除し、実行時無限ループする場合がある不具合を解決しました。

＜例＞

```
int x;
extern void f1();
extern void f2();
void f() {
    if (x == 2){
        f1();          // then 節が関数呼び出しで終わっている
    }
    f2();              //  関数の最後の処理が関数呼び出し
    return;
}

_f:
            MOV.L       L14,R6          ; _x
            MOV.L       @R6,R0
            CMP/EQ      #2,R0
            BT          L11
L12:
            MOV.L       L14+4,R2        ; _f2
            JMP         @R2             ;  関数の最後が関数呼び出しのため、JMP 命令に
            NOP                         ;  変換し、RTS  を削除
L11:
            MOV.L        L14+8,R2       ; _f1
            JSR         @R2             ;  ここに関数呼び出し命令があるにも関わらず
            NOP                         ; PR レジスタの退避回復が行われていない
            BRA         L12
            NOP
```

【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) speed オプションを指定している。

(2) 関数の最後の処理が関数呼び出しである。

(3) (2)の関数呼び出しの直前に if-then 節があり、then 節の最後が関数呼び出しである。

(4) (2)の関数呼び出しがインライン展開されない。

3.5 T ビットの不当参照 (Ver.7.0.04 -> Ver.7.0.06)

以下のケースにおいて、不当に条件分岐を行う場合がある不具合を解決しました。

(1) 以下 C プログラムをコンパイルした場合

＜例＞

```
#include <machine.h>
extern void f();
int a;
void func() {
    int b;
    b = (a == 0);          // 比較結果を変数に格納 (A)
    f();                   // 関数呼び出しまたは組み込み関数 set_cr()
    if (b) {               // (A)の変数を 0/1 比較
        a=1;
    }
}
```

```
_func:
        STS.L     PR,@-R15
        MOV.L     L13,R6        ; _a
        MOV.L     @R6,R2
        TST       R2,R2         ; 比較結果を T ビットに格納
        MOV.L     L13+4,R2      ; _f
        JSR       @R2           ; 関数呼び出し先で T ビットが
        NOP                     ; 破壊される可能性あり
        BF        L12           ; T ビットを参照し分岐
        MOV.L     L13,R6        ; _a
        MOV       #1,R2
        MOV.L     R2,@R6
L12:
        LDS.L     @R15+,PR
        RTS
        NOP
```

(2) C++プログラムで、関数内のローカルブロック内で宣言されたデストラクタ呼び出しのあるクラスを記述した場合

【発生条件】

以下の(1)～(3)または(4)～(5)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 を指定する。

(2) C プログラムで比較結果を変数に格納している。

(3) (2)の変数を関数呼び出しまたは組み込み関数 set_cr()使用後に 0/1 比較している。

または

(4) optimize=1 を指定する。

(5) C++プログラムで関数内のローカルブロック内で宣言されたデストラクタ呼び出しのあるクラスを記述している。

3.6 定数値の不当共通化 (Ver.7.0.04 -> Ver.7.0.06)

以下 C ソースを optimize=1 でコンパイル時、定数値を不当に共通化し、参照時の値が実際と異なる場合がある不具合を解決しました。

＜例＞

```
#define a (*(volatile unsigned short *)0x400)
#define b (*(volatile unsigned short *)0x4000)
#define c (*(volatile unsigned short *)0x402)
int d;
void func() {
    a = 0x8000;         /* (A)    */
    b = 0x8000;         /* (A')   */
    d = c + 0x8000;     /* (B)    */
}

_func:
            MOV.W       L15,R6          ; H'8000    R6 に 0xFFFF8000 を設定
            MOV         #4,R5
            MOV         #64,R2
            SHLL8       R5
            SHLL8       R2
            MOV.W       R6,@R5          ; (A)    a に 0x8000 を設定
            MOV.W       R6,@R2          ; (A') b に 0x8000 を設定
            MOV.W       @(2,R5),R0
            EXTU.W      R0,R2
            ADD         R6,R2           ; R2 に c+0xFFFF8000 の結果を設定
            MOV         R2,R6
            MOV.L       L15+4,R2        ; _d
            RTS
            MOV.L       R6,@R2          ; (B) d に c+0xFFFF8000 の結果を格納し NG
```

【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 を指定する。

(2) 関数内で 128～255 もしくは 32768～65535 の範囲の同じ定数値を複数回使用している。

(3) (2)の定数値が無符号で、異なるサイズで使用される。

上記例では(A)(A')では 2 バイト、(B)では 4 バイト

3.7 リテラルプール出力位置不正 (Ver.7.0.04 -> Ver.7.0.06)

align16 オプションを指定してコンパイルした場合、リテラルプールの参照が届かなくなる場合がある不具合を解決しました。

(1) code=machinecode を指定した場合

goptimize オプションを指定時はリンク時内部エラー、指定しない場合はオブジェクト不正になる場合がある。

(2) code=asmcode を指定した場合

アセンブル時エラーになる場合がある。

＜例＞

```
                    :
        MOV.L       L154+2,R2       ; L158   リテラル(A)を参照(1)
        MOV.L       R2,@R15
        MOV.L       L154+6,R2       ; _printf  リテラル(B)を参照(2)
        JSR         @R2
        NOP
                    :
        .ALIGN      16
L86:
        ADD         #1,R2
        BRA         L153            ; 無条件分岐を生成しリテラルを出力
        MOV.L       R2,@R4
L154:
        .RES.W      1
        .DATA.L     L158            ; (A) (1)から disp が届かない位置にリテラル出力
        .DATA.L     _printf         ; (B) (2)から disp が届かない位置にリテラル出力
        .ALIGN      16
L153:
                    :
```

【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) align16 オプションを指定する。

(2) 無条件分岐命令を生成してリテラルプールを出力する。

3.8 遅延スロットへの不当命令移動 (Ver.7.0.04 -> Ver.7.0.06)

optimize=1 を指定してコンパイルした場合、不当に遅延スロットに命令を移動し、レジスタを破壊する場合がある不具合を解決しました。

＜移動前＞

```
        :
    SHLL    R2
    MOV     R2,R0
    MOVA    L88,R0
    BRA     L144
    NOP
        :
```

＜移動後＞

```
        :
    SHLL    R2
                    ; 遅延スロットに移動
    MOVA    L88,R0  ; R0 に値を設定
    BRA     L144
    MOV     R2,R0   ; MOVA で設定した R0 を破壊
        :
```

【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) 同一レジスタに値を設定するコードが連続する。

3.9 GBR 相対論理演算生成時の offset 不正 (Ver.7.0.04 -> Ver.7.0.06)

optimize=1 を指定してコンパイルし、かつ 1 バイト構造体メンバの論理演算を GBR 相対で行うコードを生成した場合、その offset 値が不正になる場合がある不具合を解決しました。

＜例＞

```
struct {
    int a;
    unsigned char b;
} ST;
char c;
void f() {
    ST.b |= 1;
    c &= 1;
}

_f:
        STC     GBR,@-R15
        MOV     #0,R0       ; H'00000000
        LDC     R0,GBR
        MOV.L   L11+2,R0    ; H'00000008+_ST   <- 実際は(ST+4)
        OR.B    #1,@(R0,GBR)
        MOV.L   L11+6,R0    ; _c
        AND.B   #1,@(R0,GBR)
        RTS
        LDC     @R15+,GBR
```

【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 を指定する。

(2) gbr=user を指定しかつ#pragma gbr_base/gbr_base1 を使用している、または gbr=auto を指定しかつ map オプションを指定していない。

(3) サイズが 1 のメンバを含むグローバル構造体が存在する。

(4) サイズ 1 の構造体メンバは構造体先頭でない。

(5) (3)のメンバが関数内で論理演算で使用されている。

(6) (3)のメンバは論理演算以外では使用されていない。

(7) 関数内に論理演算のみで使用される外部変数が(3)以外に存在する。

3.10 SWAP 命令直後の不当ゼロ拡張 (Ver.7.0.04 -> Ver.7.0.06)

optimize=1 を指定してコンパイルした場合、組み込み関数 swapb、swapw、end_cnvl の結果代入先にポインタを指定すると、不当にゼロ拡張を行う場合がある不具合を解決しました。

＜例＞

```
#include <machine.h>
unsigned short *a,*b;
void func() {
      *b=swapb(*a);
}

_func:
            MOV.L       L13+2,R2      ; _a
            MOV.L       L13+6,R5      ; _b
            MOV.L       @R2,R6
            MOV.W       @R6,R2
            SWAP.B      R2,R6
            MOV.L       @R5,R2
            EXTU.B      R6,R6         ; SWAP 結果を不正にゼロ拡張
            RTS
            MOV.W       R6,@R2
```

【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 を指定する。

(2) 組み込み関数 swapb、swapw、end_cnvl を使用している。

(3) (2)の組み込み関数の結果代入先がポインタである。

3.11 ビットフィールドデータ出力不正 (Ver.7.0.04 -> Ver.7.0.06)

無名ビットフィールドを含む構造体に初期値を設定した場合、その初期値が不正になる場合がある不具合を解決しました。

＜例＞

```
struct st {
    short a:4;
    short b;
    short :12;        // 無名ビットフィールド
    short c:4;
} ST={1,1,3};

_ST:
            .DATA.W     H'1000
            .DATA.W     H'0001
            .DATAB.B    1,0             ; 正しくは
            .DATA.W     H'0300          ; .DATA.W H'0003
```

【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) 構造体の中にビットフィールド、無名ビットフィールド、ビットフィールドでないメンバが混在し、以下の順番で定義されている。

```
struct A{
            :
    ビットフィールド
            :
    ビットフィールドでないメンバ
    無名ビットフィールド        // (A)
    ビットフィールド            // (B)
            :
};
```

(2) (A)(B)のメンバの型サイズが 2 バイト以上である。

(3) (A)の無名ビットフィールドのサイズが 8 ビット以上である。

(4) (A)(B)のビットフィールドのサイズの合計が以下の通りである。

(a) (A)(B)の型サイズがともに 2 バイトの場合　：16 ビット以下

(b) (A)(B)の型サイズがともに 4 バイトの場合　：32 ビット以下

(5) 構造体の変数が定義時に初期値設定されている。

3.12 ループ展開不正 (Ver.7.0.04 -> Ver.7.0.06)

speed オプションまたは loop オプション指定時、ループ展開最適化でループ判定式を不当に置きかえる場合がある不具合を解決しました。

＜例＞

```
int a[100];
void main(int n) {
    int i;
    for (i=0; i<n; i++) {
        a[i] = 0;
    }
}

_main:
        MOV     R4,R7
        ADD     #-1,R4          ; R4 が 0x80000000 の場合、アンダフローとなり
                                ; R4 は 0x7FFFFFFF となる
        MOV     R4,R6
        CMP/PL  R4              ; 比較結果が実際の結果と逆になる
        MOV     #0,R4
        BF      L12
        ADD     #-1,R6
              :
```

【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) speed オプションまたは loop オプションを指定している。

(2) 関数内でループ文を使用している。

(3) ループ上限値が以下の通りである。

(a) 制御変数がインクリメントされていくループの場合(i += step)

ループ上限値が 0x80000000～0x80000000+step-1 の範囲である。

(b) 制御変数がデクリメントされていくループの場合(i -= step)

ループ上限値が 0x7FFFFFFF～0x7FFFFFFF-step+1 の範囲である。

上限値が変数の場合、変数に(a)または(b)を満たす値が設定されている場合に動作不正になる。

3.13 構造体、配列引数の不当参照 (Ver.7.0.04 -> Ver.7.0.06)

関数の引数に構造体/共用体または配列を使用した場合、そのメンバや配列要素を参照する時に、不正なアドレスを参照する場合がある不具合を対策しました。

＜例＞

```
typedef struct{
    int A[10];
    double B;
    char F[20];
} ST;
extern ST f(ST a,ST b);
ST S;
extern int X;
void func(ST a,ST b) {
    ST t;
    if (a.B!=f(S,t).B){
        X++;
    }
    if (a.B!=b.B){
        X++;
    }
}
```

```
                    :
L12:
            MOV     R15,R2
            MOV.W   L15+2,R0    ; H'014C
            ADD     R0,R2
            MOV     R15,R0
            MOV.W   L15+4,R0    ; H'0190    R0 を不正に破壊
            ADD     R0,R0
            MOV.L   @R2,R4
            MOV.L   @(4,R2),R7
            MOV     R0,R2
            MOV.L   @R2,R6      ; b.B でない場所をアクセス
                    :
```

【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) 引数が構造体/共用体または配列である関数が存在する。

(2) 該当関数内で引数の構造体/共用体メンバまたは配列要素を参照している。

3.14 JMP 命令の不当削除 (Ver.7.0.04 -> Ver.7.0.06)

speed オプションを指定してコンパイルした場合、JMP 命令を不当に削除する場合がある不具合を対策しました。

＜例＞

```
void f(){
    int i,j=0;
    for (i=0; i<10; i++)
        if (j%2) j++;
    sub();
}

void sub() {
    :
}

_f:
                    :
L20:
            ADD     #-1,R5
            TST     R5,R5
            BF      L11
            MOV.L   L23,R2      ; _sub   以下 3 命令が削除される
            JMP     @R2         ;                    ↓
            NOP                 ;
L18:
            MOV     R6,R0
            AND     #1,R0
            BRA     L16
            MOV     R0,R2
L13:
            MOV     R6,R0
            AND     #1,R0
            BRA     L14
            MOV     R0,R2
_sub:
                    :
```

【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) speed オプションを指定する。

(2) 該当関数の最後の処理が関数呼び出しである。

(3) (2)の関数呼び出しがインライン展開されない。

(4) (2)の関数呼び出し先が該当関数の直下にある。

(5) 該当関数内でループ文や条件文を使用している。

3.15 ループ判定式不正 (Ver.7.0.04 -> Ver.7.0.06)

以下 C ソースを optimize=1 でコンパイル時、ループ判定式を不当に置きかえる場合がある不具合を対策しました。

＜例＞

```
void f1() {
    int i;
    for (i=-1; i<INT_MAX; i++) {
        a[i] = 0;
    }
}

_f1:
        MOV.L   L13,R2      ; _a
        MOV     #-4,R6      ; H'FFFFFFFC
        MOV     R6,R5
        MOV     #0,R4       ; H'00000000
        ADD     #-4,R2
L11:
        ADD     #4,R6
        MOV.L   R4,@R2
        CMP/GE  R5,R6       ; H'FFFFFFFC と比較
        ADD     #4,R2
        BF      L11
        RTS
        NOP
```

【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) 関数内でループ文を使用している。

(3) ループ上限値 - ループ下限値がオーバフローする。

例えば、for (i=-1; i < 0x7FFFFFFF; i++)

3.16 スタックアクセス不正 (Ver.7.0.06 -> Ver.7.1.00)

レジスタ渡しパラメタのアドレスを参照すると、スタック上のデータを不正に上書きする場合がある不具合を解決しました。

＜例＞

```
extern void er();
extern char f2(char *a);
void f(char a) {
    char c;
    a++;                    // レジスタ渡しパラメタに代入
    do {
        c=f2(&a);           // レジスタ渡しパラメタのアドレスを参照
        if (c) er();
        else return;
    } while(c);
}

_f:
            MOV.L       R13,@-R15
            MOV.L       R14,@-R15
            STS.L       PR,@-R15
            ADD         #-4,R15
            MOV         R4,R0
            ADD         #1,R0
            MOV.B       R0,@(3,R15)     ; ※
            MOV.L       R4,@R15         ; ※で設定した a の値を上書き
            MOV.L       L14+2,R13       ; _f2
                    :
```

【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) レジスタ渡しパラメタが存在する。
(3) 当該パラメタが関数内でアドレス参照される。
(4) 当該パラメタのアドレス参照より前に、当該パラメタへの代入がある。
(5) スケジューリング最適化により、レジスタパラメタのスタックへのコピー命令が当該パラメタへの代入命令より前に移動される。

3.17 #pragma global_register 指定レジスタの代入削除不正 (Ver.7.0.06 -> Ver.7.1.00)

#pragma global_register 指定したレジスタの代入を不正に削除する場合がある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) #pragma global_register を指定している。

(2) optimize=1 を指定している。

(3) (1)で指定した変数が複合代入式などで 1 つの式で定義・使用される。

＜例＞

```
#pragma global_register (a=R14)
                :
a += b;      // もしくは a=a+b;
```

3.18 32bit ビットフィールド比較不正 (Ver.7.0.06 -> Ver.7.1.00)

32bit ビットフィールドを 0 と比較した場合、比較前に不正な 0 との AND が生成され、判定結果が常に真(または偽)となる場合がある不具合を解決しました。

＜例＞

```
struct ST {
    unsigned int b: 32;
};

void f(struct ST *x) {
    if (x->b) {
            :
    }
}

            :
    MOV.L       @R4,R0
    AND         #0,R0       ; <- 0 と AND をとる
    TST         R0,R0       ; 必ず真となる
    BF          L12         ; L12 への分岐は起きない
            :
```

【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) 構造体に 32bit ビットフィールドメンバが存在する。

(2) 当該メンバと 0 の比較(==もしくは!=)を行っている。

3.19 trapa_svc 使用時のスタック移動不正 (Ver.7.0.06 -> Ver.7.1.00)

pic=1 を指定してコンパイルした場合、組み込み関数 trapa_svc を使用している関数のアドレスロード時に、不当にスタック移動命令を出力する場合がある不具合を解決しました。

＜例＞

```
#include <machine.h>
extern char *b(void (*yyy)(char));

void y(char c) {
        trapa_svc(160, 10, c);
}

char *a(void) {
    return b(y);
}

_a:
        MOV.L   L14,R4          ; _y-L12
        MOVA    L12,R0
        ADD     R0,R4
L12:
        MOV.L   @R15+,R0        ; <-不要なスタック移動命令
        MOV.L   L14+4,R2        ; _b-L13
        MOVA    L13,R0
        ADD     R0,R2
L13:
        JMP     @R2
        MOV.L   @R15+,R0        ; <-不要なスタック移動命令
```

【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=sh1 以外を指定している。
(2) pic=1 を指定している。
(3) 組み込み関数 trapa_svc を使用している。
(4) trapa_svc を使用している関数のアドレスロード位置が、trapa_svc の呼び出し位置より後に出現する。

3.20 R0-R7 へのコピー命令の不当移動 (Ver.7.0.06 -> Ver.7.1.00)

最適化により、R0-R7 へのコピー命令もしくは拡張命令が関数呼び出しを超えて不正に移動し、 CMP/EQ(TST)の結果が不正となる場合がある不具合を解決しました。

＜例＞

```
        :
MOV.L   @R4,R2
MOV     R5,R14
MOV     #1,R5       ; H'00000001
ADD     #24,R2
MOV.L   @R2,R6
EXTU.W  R14,R1      ; R1 の定義が JSR を越えて移動
MOV.L   @(8,R2),R7
JSR     @R7         ; 呼び出し先で R1 を破壊する場合あり
ADD     R6,R4
MOV     #4,R2       ; H'00000004
CMP/EQ  R2,R1       ; 破壊された R1 の値を使って比較
EXTU.W  R0,R0
BF      L16
        :
```

【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) 関数内に条件分岐と、関数呼び出しがある。

(3) 以下最適化を実施。

＜最適化前＞

```
MOV     R0, Rn      ; または EXTU   R0,Rn
        :
MOV     Rx, R0      ; または EXTU   Rx,R0
TST     #imm, R0    ; または CMP/EQ   #imm,R0
MOV     Rn, R0      ; または  EXTU   Rn,R0
```

＜最適化後＞

```
MOV     Rx, Rm      ; または EXTU   Rx,Rm
MOV     #imm, Ry
TST     Ry, Rm      ; または CMP/EQ   Ry,Rm
```

(4) (3)の最適化で、Rm レジスタが R0-R7 で当該関数で他に使用されていない。

(5) (3)の最適化で生成した MOV Rx,Rm(または EXTU)命令が他の最適化により、関数呼び出しを超えて移動される。

3.21 SH-2E FDIV チップ不具合対策 (Ver.7.0.06 -> Ver.7.1.00)

SH7055RF FPU FDIV 関連不具合(TN-SH7-402A)を対策しました。cpu=sh2e かつ patch=7055 オプション指定時にチップ不具合を回避したオブジェクトを出力します。

4. 標準ライブラリ構築ツール

4.1 realloc メモリ確保サイズ (Ver.2.0 -> Ver2.0(01))

realloc をしていると、まだメモリが十分に余っているはずなのに malloc で確保できなくなる場合がある問題を解決しました。

5. フォーマットコンバータ

5.1 圧縮フォルダ内のファイルの入力 (Ver.1.0C -> Ver.1.0.04)

圧縮フォルダ内のファイルを入力しようとした場合、ファイルを認識できない(G3002 エラー)不具合を解決しました。

6. 最適化リンケージエディタ

6.1 output オプションでアドレス範囲指定時の内部エラー(7041) (Ver.7.1.04 -> Ver7.1.05)

output オプションを指定した際に、アドレス範囲で出力指定した場合に内部エラー(7041)となる場合がある不具合を解決しました。

6.2 最適化抑止指定時の不当な最適化 (Ver.7.1.04 -> Ver7.1.05)

下記条件を全て満たした際に、最適化抑止範囲が有効にならない不具合を解決しました。

(1) goptimize オプション指定ありのファイルを入力
(2) optlnk で最適化を指定
(3) absolute_forbid オプションを指定
(4) 最適化抑止範囲を二つ以上記述(二つ目以降の指定が無効となる)

6.3 relocate ファイル生成時のオブジェクト不正 (Ver.7.1.04 -> Ver7.1.05)

下記条件を全て満たした際に、不正なオブジェクトコードが生成される不具合を解決しました。

(1) 入力に relocate ファイルを指定
(2) 出力に relocate ファイルを指定
(3) delete,もしくは rename オプションを指定

6.4 map 最適化使用時の不当なエラー(L2410)出力 (Ver.7.1.04 -> Ver7.1.05)

map 最適化指定時に、オーバーレイを使用している場合に不当なエラーが出力される不具合を解決しました。

6.5 レジスタ退避/回復コード最適化指定時のインターナルエラー(1703) (Ver.7.1.04 -> Ver7.1.05)

レジスタ退避/回復コード最適化を指定時に、インターナルエラー(1703)となる場合がある不具合を解決しました。

6.6 最適化指定時の内部エラー (Ver.7.1.05 -> Ver7.1.06)

最適化指定時に、内部エラー(1703,1704)が発生する場合がある不具合を解決しました。

6.7 output オプション指定時の内部エラー (Ver.7.1.05 -> Ver7.1.06)

output オプション指定時に、出力範囲をアドレスで指定した場合に内部エラー(7707)となる場合がある不具合を解決しました。

6.8 HEX,BIN,Stype 出力の内部エラー (Ver.7.1.05 -> Ver7.1.06)

C++にてソースを記述した場合、出力形式が HEX,BIN,Stype の場合に内部エラー(3304)となる場合がある不具合を解決しました。

6.9 map オプション指定時の内部エラー (Ver.7.1.05 -> Ver7.1.06)

map オプション指定時に、内部エラー(8093)となる場合がある不具合を解決しました。

6.10 レジスタ退避/回復最適化指定時の Object 不正 (Ver.7.1.05 -> Ver7.1.06)

レジスタ退避/回復コード最適化を指定した際に、下記条件のいずれかを満たす場合、不正なオブジェクトコードが出力される場合がある不具合を修正しました。

(1) ファイル内の最後の関数にリテラルが存在しない。
(2) レジスタ回復コード列の直後に分岐先が存在する。
(3) 最適化対象関数に SUBC 命令が含まれており、その SUBC 命令自体がレジスタ番号の変換対象である。

7. SuperH RISC engine シミュレータ・デバッガ (UNIX 版)

7.1 SH3-DSP ASIC コアシミュレータのサポート (Ver. 4.11 -> Ver. 4.2.00)

SH3-DSP ASIC コアシミュレータをサポートしました。

SH3-DSP ASIC コアシミュレータは協調検証をサポートしています。

8. スタック解析ツール

8.1 データマージ機能追加 (Ver.1.2.00→Ver.1.3.00)

ユーティリティツールの一つ、スタック解析ツールにデータマージ機能を追加しました。

これにより、既に作成済のアセンブリプログラム等のスタックデータと最適化リンケージエディタが生成したスタック情報ファイルのデータを合成できるようになりました。

以上