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

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

SuperH RISC engine C/C++ コンパイラパッケージ

アプリケーションノート：＜コンパイラ活用ガイド＞FPU 活用 編

本ドキュメントでは、SuperH RISC engine C/C++ コンパイラ V.9 における FPU(SH-2E, SH2A-FPU，SH-4，SH-4A)の使用方法、注意点を説明します。

誤記に関するお詫び:
本資料の13ページに誤記があり、訂正いたしました。

目次

1. 浮動小数点ユニット(FPU)について

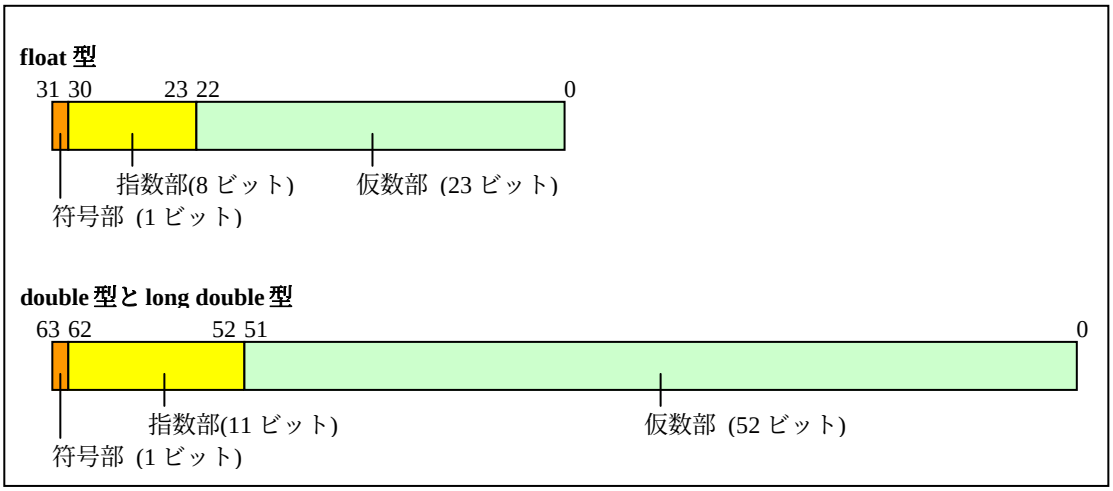
SH-2E、SH2A-FPU、SH-4、SH-4A には浮動小数点演算を高速に行う FPU が内蔵されています。SH2A-FPU、SH-4、SH-4A では倍精度(double 型)、単精度(float 型)の演算を、SH-2E では単精度の演算を FPU で行うことができます。
FPU には次のような特長があります。

- 2 つの丸めモード：近傍への丸め(Round to Nearest)および 0 方向への丸め(Round to Zero)を選択可能
  （SH-2E は Round to Zero のみ）
- 2 つの非正規化数処理モード：非正規化数の処理方法を選択可能 (SH-4, SH-4A のみ)
- 6 つの例外要因： 例外要因ごとに例外発生のマスク、イネーブルの選択が可能。
  - FPU エラー、無効演算、0 による除算、オーバフロー、アンダフロー、不正確
    (SH-2E は無効演算、0 による除算のみ)

1.1 浮動小数点数の仕様

1.1.1 浮動小数点数の内部表現

- 内部表現の形式
  float 型は IEEE の単精度形式(32 ビット)、double 型と long double 型は IEEE の倍精度形式(64 ビット)で表現します。
- 内部表現の構成
  float 型およびdouble 型とlong double 型の内部表現の構成を図 1-1に示します。



【注】 double=float を指定した場合、double 型は float 型と同じ内部表現となります。
cpu=sh2afpu|sh4|sh4a かつ fpu=single を指定した場合、double 型、long double 型は float 型と同じ内部表現となります。
cpu=sh2afpu|sh4|sh4a かつ fpu=double を指定した場合、float 型は double 型と同じ内部表現になります。

図 1-1

内部表現の各構成要素の意味を以下に示します。

(i) 符号部
浮動小数点数の符号を示します。0 の時は正、1 の時は負を示します。

(ii) 指数部
浮動小数点数の指数を 2 のべき乗で示します。

(iii) 仮数部
浮動小数点数の有効数字に対応するデータです。

- 表現する値の種類
  浮動小数点数は、通常の実数値のほかに、無限大等の値も表現することができます。浮動小数点数が表現する値の種類を以下に示します。
  (i) 正規化

指数部が 0 または全ビット 1 ではない場合です。通常の実数値を表現します。

(ii) 非正規化数
指数部が 0 で、仮数部が 0 でない場合です。絶対値の小さな実数値を表現します。

(iii) ゼロ
指数部および仮数部が 0 の場合です。値 0.0 を表現します。

(iv) 無限大
指数部が全ビット 1 で仮数部が 0 の場合です。無限大を表現します。

(v) 非数
指数部が全ビット 1 で仮数部が 0 でない場合です。「0.0/0.0」、「∞/∞」、「∞-∞」等、結果が数値に対応しない演算の結果として得られます。

浮動小数点数の表現する値を決定する条件を表 1-1に示します。

表 1-1浮動小数点数の表現する値の種類

仮数部 ＼ 指数部	0	0 でも全ビット 1 でもない	全ビット 1
0	0	正規化数	無限大
0 以外	非正規化数	正規化数	非数

【注】 非正規化数は、正規化数で表現できない範囲の絶対値の小さな浮動小数点数を表現しますが、正規化数と比較して有効桁数が少なくなっています。したがって、演算の結果あるいは途中結果が非正規化数となる場合、結果の有効桁数は保証しません。

1.1.2 単精度(float 型)のデータ形式

float 型の内部表現は、1 ビットの符号部、8 ビットの指数部、23 ビットの仮数部からなります。

- 正規化数
  符号部は、0(正)または 1(負)で、値の符号を示します。
  指数部は、1～254(2^8–2)の値をとります。実際の指数は、この値から 127 を引いた値で、その範囲は–126～127 です。
  仮数部は、0～2^{23}–1 の値をとります。実際の仮数は、2^{23}のビットを 1 と仮定し、その直後に小数点があるものとして解釈します。
  正規化数の表現する値は、

$$(-1)^{<符号部>} \times 2^{<指数部>-127} \times (1 + <仮数部> \times 2^{-23})$$

  となります。
  例:

31	30 … 23	22 … 0
1	10000000	11000000000000000000000

  符号： －
  指数： 10000000 (2) －127＝1
  仮数： 1.11 (2) ＝1.75
  値 ： －1.75×2^1＝－3.5
  ※ (2) は 2 進数を意味します。

- 非正規化数
  符号部は 0(正)または 1(負)で、値の符号を示します。
  指数部は 0 で、実際の指数は–126 になります。
  仮数部は、1～2^{23}–1 で、実際の仮数は、2^{23}のビットを 0 と仮定し、その直後に小数点があるものとして解釈します。
  非正規化数の表現する値は、

$$(-1)^{<符号部>} \times 2^{-126} \times (<仮数部> \times 2^{-23})$$

  となります。
  例:

31	30 … 23	22 … 0
0	00000000	11000000000000000000000

符号： ＋
指数： －126
仮数： 0.11 (2) ＝0.75
値 ： 0.75×2^{-126}
※ (2) は 2 進数を意味します。

- ゼロ
  符号部は 0(正)または 1(負)で、それぞれ+0.0、–0.0 を示します。
  指数部、仮数部はともに 0 です。
  +0.0、–0.0 は、ともに値としては 0.0 を示します。

- 無限大
  符号部は 0(正)または 1(負)で、それぞれ+∞、–∞を示します。
  指数部は 255(2^8–1)です。
  仮数部は 0 です。

- 非数
  指数部は 255(2^8–1)です。
  仮数部は 0 以外の値です。

【注】 CPU が SH-2E、SH2A-FPU、SH-4、SH-4A の場合、仮数部の最上位ビットが 0 の非数を qNaN、仮数部の最上位ビットが 1 の非数を sNaN と呼びます。
その他の仮数フィールドの値、および符号部については規定していません。

1.1.3 倍精度(double 型)のデータ形式

double 型と long double 型の内部表現は、1 ビットの符号部、11 ビットの指数部、52 ビットの仮数部からなります。

- 正規化数
  符号部は、0(正)または 1(負)で、値の符号を示します。
  指数部は、1～2046(2^{11}–2)の値をとります。実際の指数は、この値から 1023 を引いた値で、その範囲は–1022～1023 です。
  仮数部は、0～2^{52}–1 の値をとります。実際の仮数は、2^{52}のビットを 1 と仮定し、その直後に小数点があるものとして解釈します。
  正規化数の表現する値は、
  $(-1)^{<符号部>} \times 2^{<指数部>-1023} \times (1 + <仮数部> \times 2^{-52})$
  となります。
  例:

63	62 … 52	51 … 0
0	01111111111	11100000000000000000000000000000000000000000000000000

符号： ＋
指数： 1111111111 (2) －1023＝0
仮数： 1.111 (2) ＝1.875
値 ： $1.875 \times 2^0 = 1.875$
※ (2) は 2 進数を意味します。

- 非正規化数
  符号部は 0(正)または 1(負)で、値の符号を示します。
  指数部は 0 で、実際の指数は–1022 になります。
  仮数部は、1～2^{52}–1 で、実際の仮数は、2^{52}のビットを 0 と仮定し、その直後に小数点があるものとして解釈します。
  非正規化数の表現する値は、
  $(-1)^{<符号部>} \times 2^{-1022} \times (<仮数部> \times 2^{-23})$
  となります。
  例:

63	62 … 52	51 … 0
1	00000000000	11100000000000000000000000000000000000000000000000000

符号： －
指数： －1022
仮数： 0.111 (2) ＝0.875
値 ： 0.875×2^{-1022}
※ (2) は 2 進数を意味します。

- ゼロ
  符号部は 0(正)または 1(負)で、それぞれ+0.0、–0.0 を示します。
  指数部、仮数部はともに 0 です。
  +0.0、–0.0 は、ともに値としては 0.0 を示します。

- 無限大
  符号部は 0(正)または 1(負)で、それぞれ+∞、–∞を示します。
  指数部は 2047(2^{11}–1)です。
  仮数部は 0 です。

- 非数
  指数部は 2047(2^{11}–1)です。
  仮数部は 0 以外の値です。

【注】 CPU が SH2A-FPU、SH-4、SH-4A の場合、仮数部の最上位ビットが 0 の非数を qNaN、
仮数部の最上位ビットが 1 の非数を sNaN と呼びます。
その他の仮数フィールドの値、および符号部については規定していません。

1.2 レジスタ

1.2.1 浮動小数点ステータス／コントロールレジスタ(FPSCR)

FPSCR は、32 ビットのレジスタで、丸めモード、漸近的なアンダフロー（非正規化数）、および FPU 例外に関する詳細情報の格納を制御します。

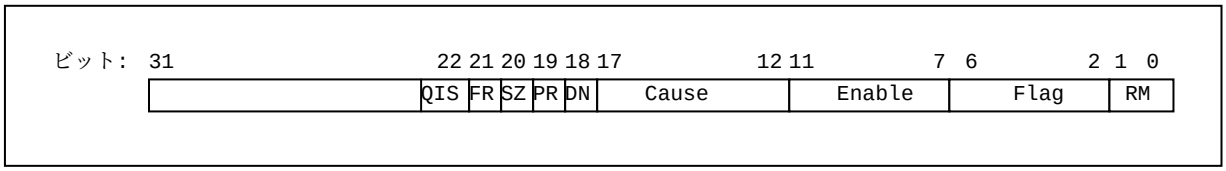


図 1-2

- QIS: (SH2A-FPU)
  qNaN あるいは±∞を sNaN として扱う。FPSCR の Enable が V=1 の時のみ有効。
  QIS=0: qNaN あるいは±∞として処理。
  QIS=1: 例外発生（sNaN と同様に処理）。
- FR: 浮動小数点レジスタバンク (SH-4、SH-4A)
  浮動小数点レジスタの FPR0_BANK0～FPR15_BANK0 と FPR0_BANK1～FPR15_BANK1 とで割り当てを切り替えます。
- SZ: 転送サイズモード (SH2A-FPU、SH-4、SH-4A)
  SZ=0: FMOV 命令のデータサイズは 32 ビットです。
  SZ=1: FMOV 命令のデータサイズは 32 ビットペア（64 ビット）です。
- PR: 精度モード (SH2A-FPU、SH-4、SH-4A)
  PR=0: 浮動小数点命令を単精度演算として実行します。
  PR=1: 浮動小数点命令を倍精度演算として実行します（倍精度がサポートされていない命令の結果は未定義です）。

[注意]
SH-4 では SZ と PR は同時に 1 にセットしないでください。この設定は予約されています [SZ, PR] = 11： 予約（FPU 演算命令は未定義です）。

- DN: 非正規化モード (SH-2E、SH2A-FPU、SH-4、SH-4A)

SH-2E、SH2A-FPU では常に DN=1。
DN=0: 非正規化数を非正規化数として扱います。
DN=1: 非正規化数を 0 として扱います。

- Cause: FPU 例外要因フィールド(SH-2E、SH2A-FPU、SH-4、SH-4A)
- Enable: FPU 例外イネーブルフィールド(SH-2E、SH2A-FPU、SH-4、SH-4A)
- Flag: FPU 例外フラグフィールド(SH-2E、SH2A-FPU、SH-4、SH-4A)

FPU 演算命令を実行すると、FPU 例外要因フィールドは最初に 0 に設定されます。
次に FPU 例外が発生すると、FPU 例外要因フィールドと FPU 例外フラグフィールドの該当ビットが 1 にセットされます。
FPU 例外フラグフィールドは、FPU 例外フラグフィールドが最後にクリアされたそれ以降に発生した例外のステータスを保持します。

表 1-2 FPU 例外処理に関連するビットの割り付け

		FPU エラー (E)	無効演算(V)	0 除算 (Z)	オーバフロー(O)	アンダフロー(U)	不正確(I)
Cause	FPU 例外要因フィールド	ビット 17	ビット 16	ビット 15	ビット 14	ビット 13	ビット 12
Enable	FPU 例外イネーブルフィールド	なし	ビット 11	ビット 10	ビット 9	ビット 8	ビット 7
Flag	FPU 例外フラグフィールド	なし	ビット 6	ビット 5	ビット 4	ビット 3	ビット 2

- RM: 丸めモード(SH-2E、SH2A-FPU、SH-4、SH-4A)

SH-2E では、常に 01 で「Round to Zero」となります。
RM=00：Round to Nearest
RM=01：Round to Zero

1.2.2 浮動小数点レジスタ

SH-2E、SH2A-FPU では 16 本、SH-4、SH-4A では 32 本の 32 ビット浮動小数点レジスタがあります。詳細を以下に示します。

- SH-2E

図 1-3に浮動小数点レジスタを示します。浮動小数点レジスタ（FRn）は 32 ビットの長さで、FR0～15 までの 16 本あります。浮動小数点レジスタは浮動小数点命令で使用します。

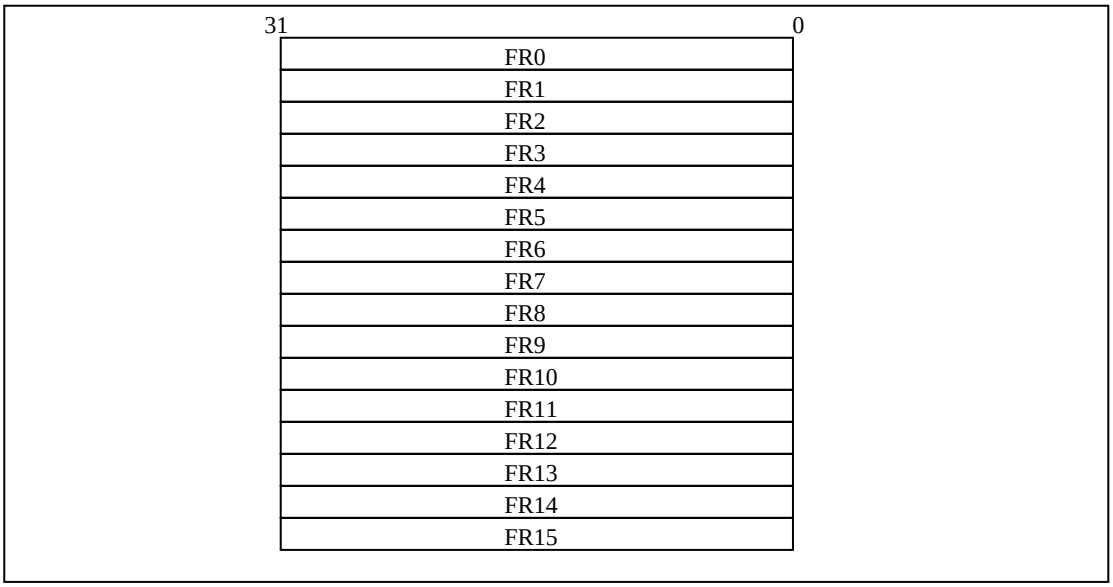


図 1-3

- SH2A-FPU
  図 1-4に浮動小数点レジスタを示します。16 本の 32 ビット浮動小数点レジスタFPR0～FPR15 があります。この 16 本のレジスタはFR0～FR15、DR0/2/4/6/8/l0/12/14 として参照されます。FPRn と参照名の対応はFPSCR の PR ビットとSZ ビットによって決まります。

  (1) 浮動小数点レジスタ　FPRn (16 レジスタ)
  FPR0, FPR l, FPR2, FPR3, FPR4, FPR5, FPR6, FPR7,FPR8, FPR9, FPR10, FPR11, FPR12, FPR13, FPR14, FPR15
  (2) 単精度浮動小数点レジスタ　FRi (16 レジスタ)
  FR0～FR15 は FPR0～FPR15 に割り当てられます。
  (3) 倍精度浮動小数点レジスタ、または単精度浮動小数点レジスタのペア　DRi (8 レジスタ)
  DR0 = {FPR0, FPR1} ,DR2 = {FPR2, FPR3 },
  DR4 = {FPR4, FPR5} , DR6 = {FPR6, FPR7} ,
  DR8 = {FPR8, FPR9}, DR10 = {FPR10, FPR11},
  DR12 = {FPR12, FPR13}, DR14 = {FPR14, FPR15}

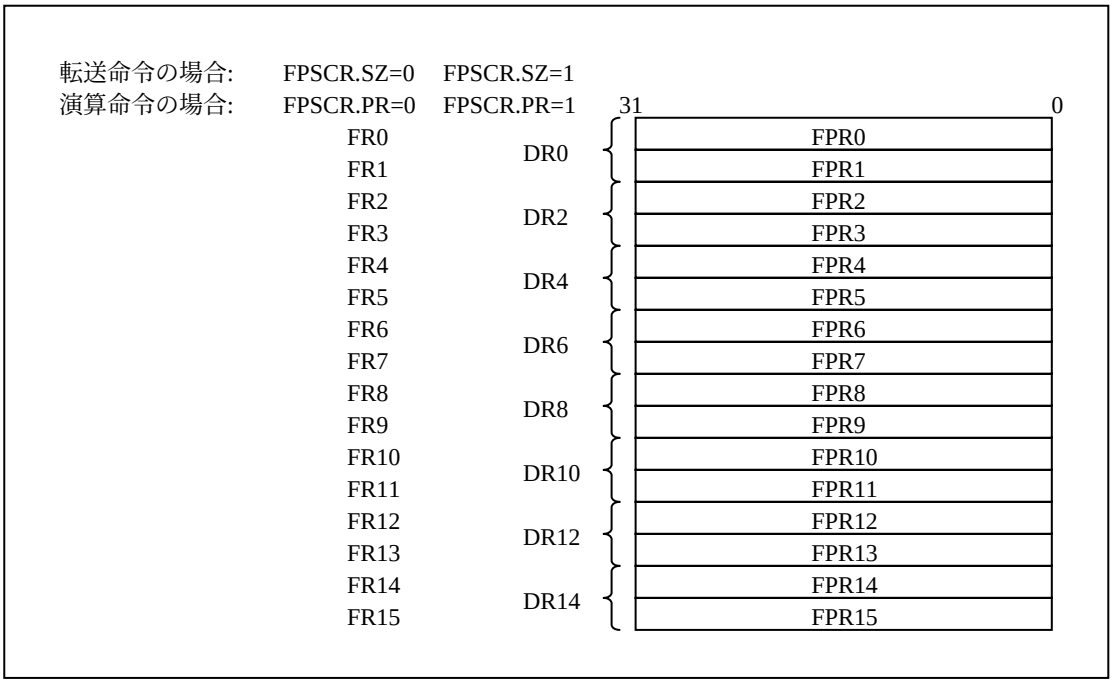


図 1-4

- SH-4・SH-4A
  図 1-5に小数点レジスタを示します。32 本の 32 ビット浮動小数点レジスタがあります。これらは、2 つのバンクで構成され、FPR0_BANK0～FPR15_BANK0、FPR0_BANK1～FPR15_BANK1 があります。また、この 32 本レジスタはFR0～FR15, DR0/2/4/6/8/l0/12/14、 FV0/4/8/12, XF0～XF15,XD0/2/4/6/8/l0/12/14、XMTRX として参照されます。FPRn_BANKi と参照名の対応はFPSCR のFR ビットによって決まります。
  (1) 浮動小数点レジスタ　FPRn_BANKi(32 レジスタ)
  FPR0_BANK0, FPR l_BANK0, FPR2_BANK0, FPR3_BANK0,
  FPR4_BANK0, FPR5_BANK0, FPR6_BANK0, FPR7_BANK0,
  FPR8_BANK0, FPR9_BANK0, FPR10_BANK0, FPR11_BANK0,
  FPR12_BANK0, FPR13_BANK0, FPR14_BANK0, FPR15_BANK0
  FPR0_BANK1 , FPR1_BANK1, FPR2_BANK1 , FPR3_BANK1,
  FPR4_BANK1, FPR5_BANK1, FPR6_BANK1, FPR7_BANK1,
  FPR8_BANK1, FPR9_BANK1, FPR10_BANK1, FPR11_BANK1,
  FPR12_BANK1 , FPR13_BANK1, FPR14_BANK1 , FPR15_BANK1
  (2) 単精度浮動小数点レジスタ FRi(16 レジスタ)
  FPSCR.FR = 0 の時は、FR0～FR15 は FPR0_BANK0～FPR15_BANK0 に割り当てられます。
  FPSCR.FR = 1 の時は、FR0～FR15 は FPR0_BANK1～FPR15_BANK1 に割り当てられます。
  (3) 倍精度浮動小数点レジスタ、または単精度浮動小数点レジスタのペア DRi(8 レジスタ)
  DR レジスタは、2 つの FR レジスタから構成されます。

DR0 = {FR0, FR1} ,DR2 = {FR2, FR3 },
DR4 = {FR4, FR5} , DR6 = {FR6, FR7} ,
DR8 = {FR8, FR9}, DR10 = {FR10, FR11},
DR12 = {FR12, FR13}, DR14 = {FR14, FR15}

(4) 単精度浮動小数点ベクトルレジスタ FVi(4 レジスタ)
FV レジスタは 4 つの FR レジスタから構成されます。
FV0 = {FR0, FR1, FR2, FR3},
FV4 = { FR4, FR5, FR6, FR7},
FV8 = {FR8, FR9, FR10, FR11},
FVl2 = {FR12, FR13, FR14, FR15}

(5) 単精度浮動小数点拡張レジスタ XFi(16 レジスタ)
FPSCR.FR = 0 の時は、XF0～XF15 は FPR0_BANK1～FPR15_BANK1 に割り当てられます。
FPSCR.FR = 1 の時は、XF0～XF15 は FPR0_BANK0～FPR15_BANK0 に割り当てられます。

(6) 単精度浮動小数点拡張レジスタのペア XDi(8 レジスタ)
XD レジスタは 2 つの XF レジスタから構成されます。
XD0 = {XF0, XF1 }, XD2 = {XF2, XF3},
XD4 = {XF4, XF5}, XD6 = {XF6, XF7},
XD8 = {XF8, XF9}, XD10 = {XF10, XF11},
XD12 = {XF12, XF13}, XD14 = {XF14, XF15}

(7) 単精度浮動小数点拡張レジスタ行列 XMTRX
XMTRX は 16 本の XF レジスタから構成されます。

$$XMTRX = \begin{pmatrix} XF0 & XF4 & XF8 & XF12 \\ XF1 & XF5 & XF9 & XF13 \\ XF2 & XF6 & XF10 & XF14 \\ XF3 & XF7 & XF11 & XF15 \end{pmatrix}$$

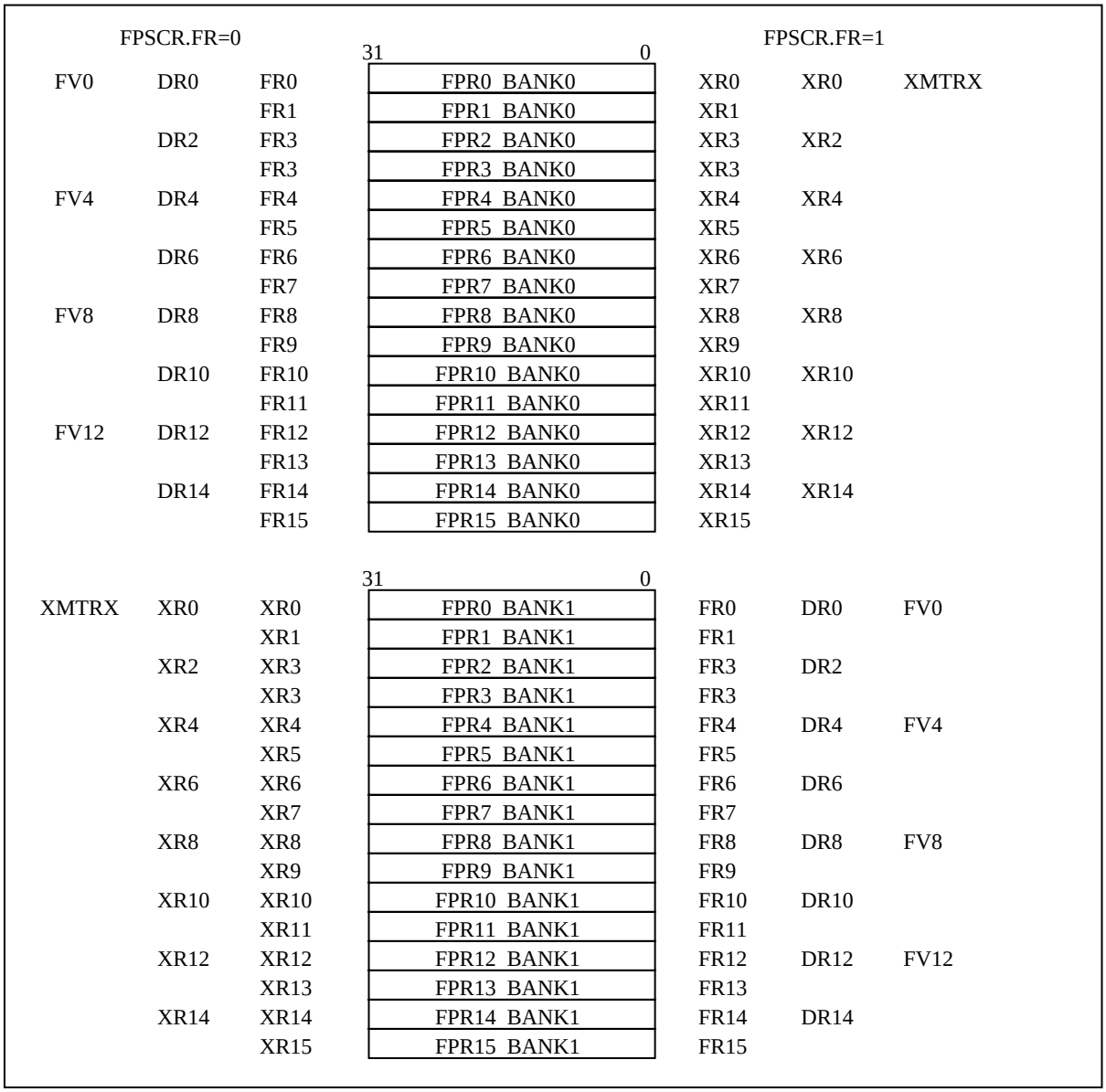

FPSCR.FR=0
31
0
FPSCR.FR=1
FV0 DR0 FR0 FPR0_BANK0 XR0 XR0 XMTRX
FR1 FPR1_BANK0 XR1
DR2 FR3 FPR2_BANK0 XR3 XR2
FR3 FPR3_BANK0 XR3
FV4 DR4 FR4 FPR4_BANK0 XR4 XR4
FR5 FPR5_BANK0 XR5
DR6 FR6 FPR6_BANK0 XR6 XR6
FR7 FPR7_BANK0 XR7
FV8 DR8 FR8 FPR8_BANK0 XR8 XR8
FR9 FPR9_BANK0 XR9
DR10 FR10 FPR10_BANK0 XR10 XR10
FR11 FPR11_BANK0 XR11
FV12 DR12 FR12 FPR12_BANK0 XR12 XR12
FR13 FPR13_BANK0 XR13
DR14 FR14 FPR14_BANK0 XR14 XR14
FR15 FPR15_BANK0 XR15
31
0
XMTRX XR0 XR0 FPR0_BANK1 FR0 DR0 FV0
XR1 FPR1_BANK1 FR1
XR2 XR3 FPR2_BANK1 FR3 DR2
XR3 FPR3_BANK1 FR3
XR4 XR4 FPR4_BANK1 FR4 DR4 FV4
XR5 FPR5_BANK1 FR5
XR6 XR6 FPR6_BANK1 FR6 DR6
XR7 FPR7_BANK1 FR7
XR8 XR8 FPR8_BANK1 FR8 DR8 FV8
XR9 FPR9_BANK1 FR9
XR10 XR10 FPR10_BANK1 FR10 DR10
XR11 FPR11_BANK1 FR11
XR12 XR12 FPR12_BANK1 FR12 DR12 FV12
XR13 FPR13_BANK1 FR13
XR14 XR14 FPR14_BANK1 FR14 DR14
XR15 FPR15_BANK1 FR15

図 1-5

1.2.3 浮動小数点通信レジスタ(FPUL)

FPU と CPU 間の情報伝達は FPUL レジスタを介して行われます。32 ビットの FPUL レジスタはシステムレジスタで、LDS、STS 命令によって CPU からもアクセスします。たとえば、汎用レジスタ R1 に格納した整数を単精度浮動小数点に変換する処理フローは次のとおりです。

R1 → （LDS 命令） → FPUL → （単精度 FLOAT 命令） → FR1

1.2.4 ステータスレジスタ(SR)

ステータスレジスタ(SR)の FD ビットが 1 の場合、FPU 命令は一般 FPU 抑止例外を発生させ、FPU 命令が遅延スロットにある場合、スロット FPU 抑止例外が発生します。（FPU 命令： H'F***命令、FPUL/FPSCR に対する LDS(.L)/STS(.L)命令）

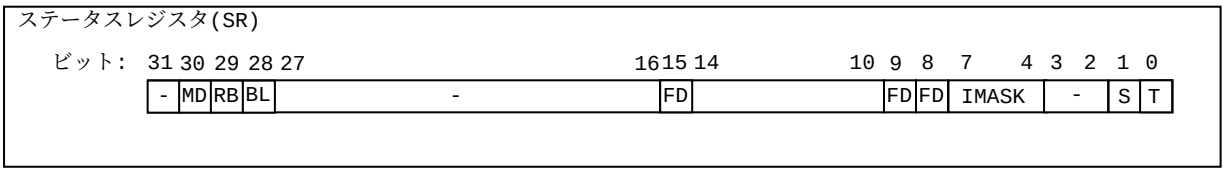


図 1-6

1.3 丸め

浮動小数点数の四則演算、定数の代入などで正確な値の内部表現の仮数が有効桁を超える場合、次の丸めを行います。

- CPU が SH-2E の場合、「Round to Zero」を行います(有効桁を超える部分を切り捨てる)。
- CPU が SH2A-FPU、SH-4 または SH-4A の場合、FPSCR の RM にて「Round to Zero」と「Round to Nearest」を選択することができます。
- FPU を持たない CPU は浮動小数点演算を実行時ルーチンで処理しており、「Round to Nearest」を行います。

[Round to Nearest について]

「Round to Nearest」では値を近似する二つの浮動小数点数の内部表現が近い方に向かって丸めます。
仮数最終桁の次の値で近似する方向を求めます。また、近似前の値がその値を近似する二つの浮動小数点数のちょうど中央である場合は、仮数の最終桁が 0 となる方向に丸めます。

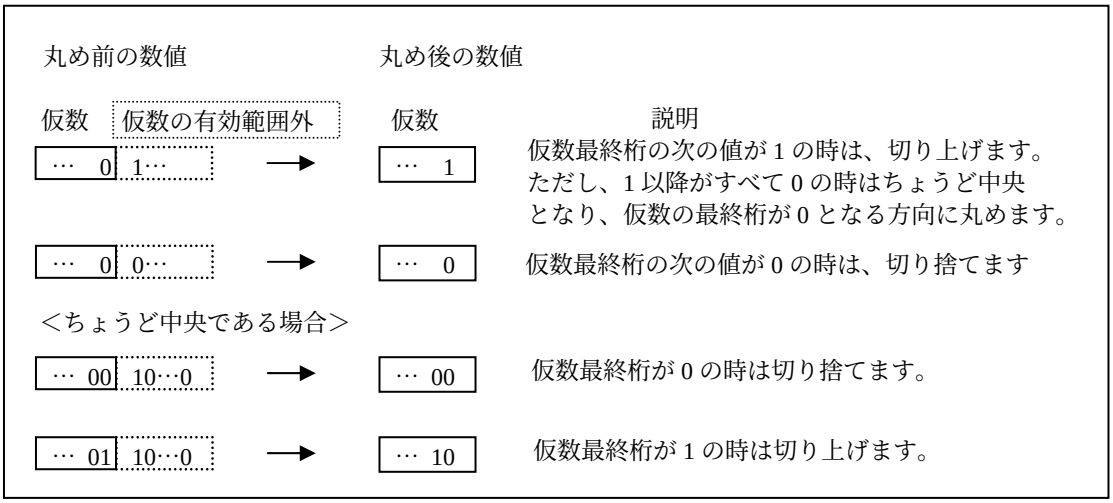


図 1-7

例えば、0.1 は単精度浮動小数点数の形式では有効桁内で正確に表現できないため丸めが行われます。
Round to Zero では「0x3DCCCCCC」、Round to Nearest では「0x3DCCCCCD」と表現されます。
この値は 0.1 に限りなく近い 0.0 999・・・・ という近似値になります。

2. 浮動小数点演算に関するオプション及び#pragma

2.1 double→float 変換(DOuble=Float) ＜SH-2E＞

プログラム内で使用している float 型、double 型の浮動小数点数(long double 型と宣言しているもの以外) を単精度として扱うためのオプションです。 SH-2E では、FPU が単精度のみをサポートしているため、倍精度の演算は実行時ルーチン呼び出しで処理されます。本オプションを指定することで、long double 型以外の浮動小数点演算を FPU で扱うことができるようになり、浮動小数点演算の速度を向上させることが出来ます。

[補足]
本オプションは、FPU が内蔵されていない CPU(SH-1、SH-2、SH-2A、SH2-DSP、SH-3、SH3-DSP、SH4AL-DSP)でも指定可能です。

例:

ソースコード

```
double func(double a, float b)
{
    return a + b;
}
```

double=float未指定時(デフォルト)のアセンブリ展開コード

```
_func:
    STS.L       PR,@-R15
    MOV         R15,R2
    ADD         #8,R2
    MOV.L       @(4,R2),R1 ; (part of)a
    MOV.L       @R2,R4     ; (part of)a
    MOV.L       R1,@-R15
    MOV.L       R4,@-R15
    ADD         #-8,R15
    MOV         R15,R4
    MOV.L       R4,@-R15
    MOV.L       L11,R5     ; __ftod_a
    JSR         @R5
    FMOV.S      FR4,FR0
    ADD         #4,R15
    MOV.L       @(20,R15),R6
    MOV.L       L11+4,R7   ; __addd_a
    JSR         @R7
    MOV.L       R6,@-R15
    ADD         #20,R15
    LDS.L       @R15+,PR
    RTS
    NOP
L11:
    .DATA.L     __ftod_a
    .DATA.L     __addd_a
```

double=float指定時のアセンブリ展開コード

```
_func:
    FADD        FR5,FR4
    RTS
    FMOV.S      FR4,FR0
```

[High-Performance Embededd-Workshop(以後、HEW と略します) でのオプション設定方法]

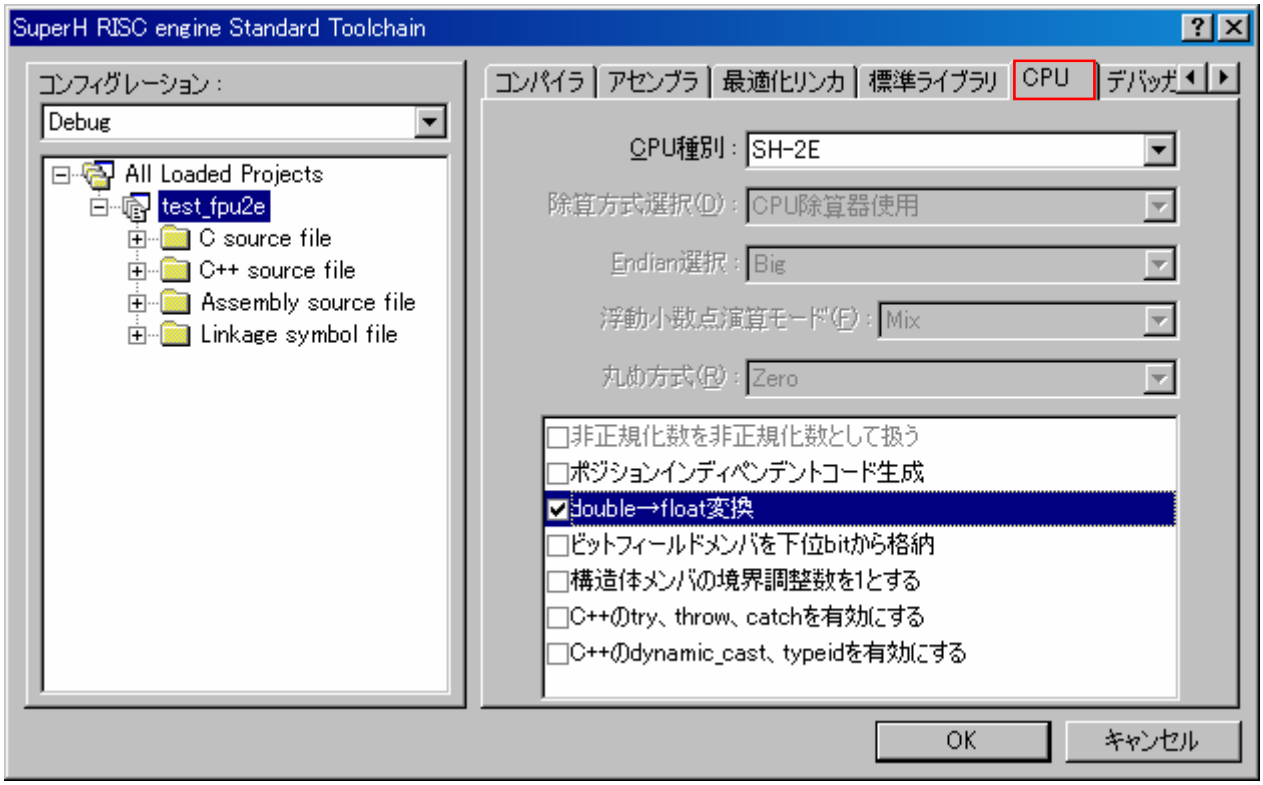


図 2-1

2.2 浮動小数点演算モード(FPu={Single|Double}) ＜SH2A-FPU、SH-4、SH-4A＞

プログラム内で使用している浮動小数点数の型を統一して扱うためのオプションです。 SH2A-FPU、SH-4、SH-4A の FPU は、単精度、倍精度のどちらの演算もサポートしていますが、違う精度の演算を行う際には、FPSCR の PR ビットの切り替えが必要となります。そのため、異なる精度の浮動小数点演算を行うと、パフォーマンスが低下する恐れがあります。 FPU オプションを用いてプログラム内の浮動小数点数の型を統一させると、パフォーマンスの向上が期待できます。

- Mix [デフォルト]
  C/C++ソースの記述どおりの型で演算を行います。
  コンパイラは FPSCR レジスタの PR ビットを切り替えるコードを生成します。
- Single(fpu=single)
  すべての浮動小数点演算を単精度浮動小数点数(float 型)で演算します。
  コンパイラは FPSCR レジスタの PR ビットを操作しません。
- Double(fpu=double)
  すべての浮動小数点演算を倍精度浮動小数点数(double 型)で演算します。
  コンパイラは FPSCR レジスタの PR ビットを操作しません。

[注意]

“Single” および “Double” を選択した場合、コンパイラは FPSCR の PR ビットを操作しないため、ユーザプログラムで初期値を設定する必要があります。

なお、CPU の初期状態では PR ビットの値は「0(=単精度)」のため、“Double”を選択した時は初期値設定を忘れると演算が不正となるので、特にご注意ください。FPSCR は組み込み関数 set_fpscr で設定できます。

例:

ソースコード
```
double func(double a, float b)
{
    return a + b;
}
```

fpu未指定時（デフォルト）のアセンブリ展開コード
```
_func:
    FLDS        FR6,FPUL
    STS         FPSCR,R2
    MOV         #8,R6       ; H'00000008
    SHLL16      R6
    OR          R6,R2
    LDS         R2,FPSCR
    FCNVSD      FPUL,DR0
    RTS
    FADD        DR4,DR0
```

fpu=single 指定時のアセンブリ展開コード
```
_func:
    FADD        FR5,FR4
    RTS
    FMOV.S      FR4,FR0
```

fpu=double指定時のアセンブリ展開コード
```
_func:
    FMOV.S      FR4,FR0
    FMOV.S      FR5,FR1
    RTS
    FADD        DR6,DR0
```

[HEW でのオプション設定方法]

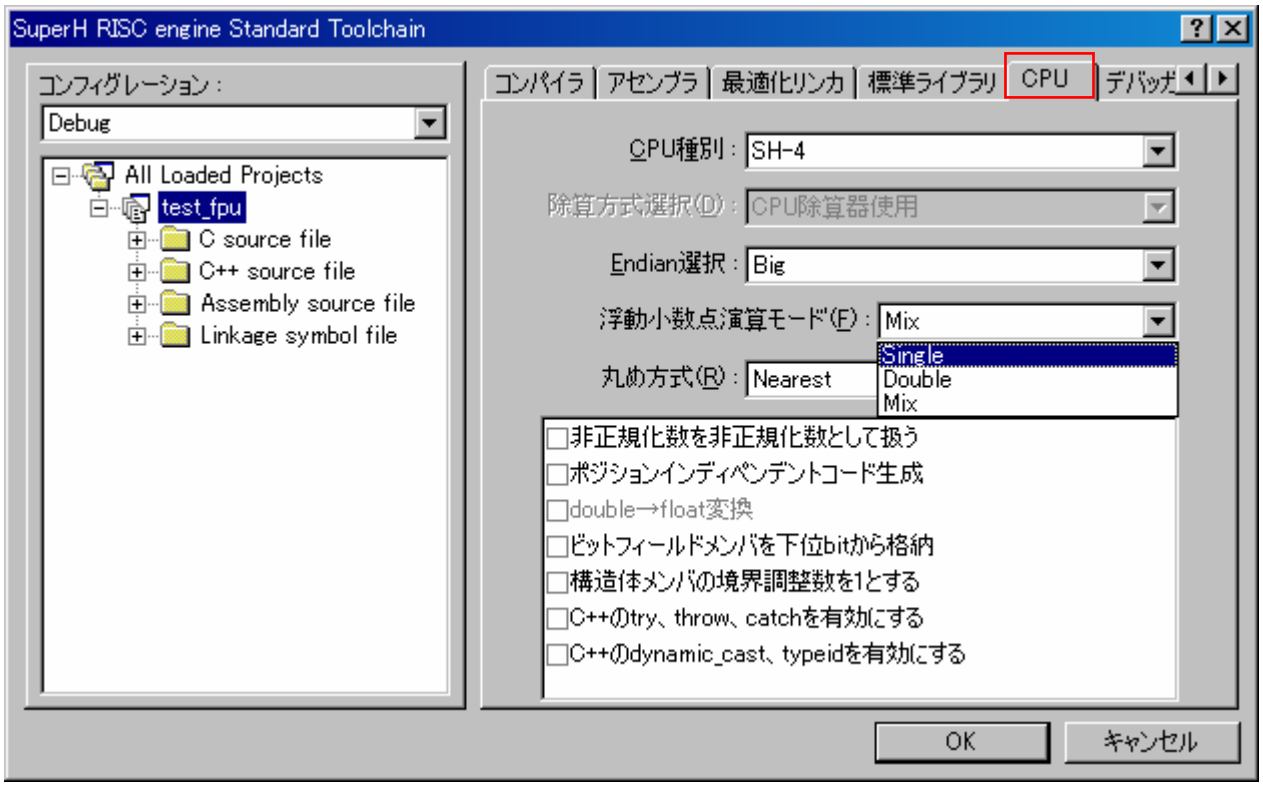


図 2-2

2.3 丸め方式(Round={Zero|Nearest}) ＜SH2A-FPU、SH-4、SH-4A＞

SH2A-FPU、SH-4、SH-4Aでは丸め方式を選択することができます。 丸め方式を設定する際には、FPSCRのRMビットの設定とコンパイラのroundオプションの設定を同じとする必要があります。 コンパイラのroundオプションは、図 2-3の[丸め方式]の選択で行います。

- Zero(round=zero) [デフォルト]
  Round to Zero で丸めます。
- Nearest(round=nearest)
  Round to Nearest で丸めます。

コンパイラは FPSCR の RM ビットの値を変更するコードは生成しませんので、RM ビットの設定はユーザプログラムにて明示的に行う必要があります。~~CPU の初期状態では RM ビットの値は「00(=Round to Zero)」です。“Nearest” を指定する場合は、RM ビットを「01(=Round to Nearest)」に設定してください。~~FPSCR は組み込み関数 set_fpscr で設定できます。

CPU の初期状態ではRM ビットの値は「01(=Round to Zero)」です。
“ Nearest ” を指定する場合は、RMビットを「00(=Round to Nearest)」に設定してください。

例:

ソースコード float ff = 0.1f; round=zero(デフォルト)指定時のアセンブリ展開コード _ff: .DATA.L H'3DCCCCCC	round=nearest 指定時のアセンブリ展開コード _ff: .DATA.L H'3DCCCCCD

[HEW でのオプション設定方法]

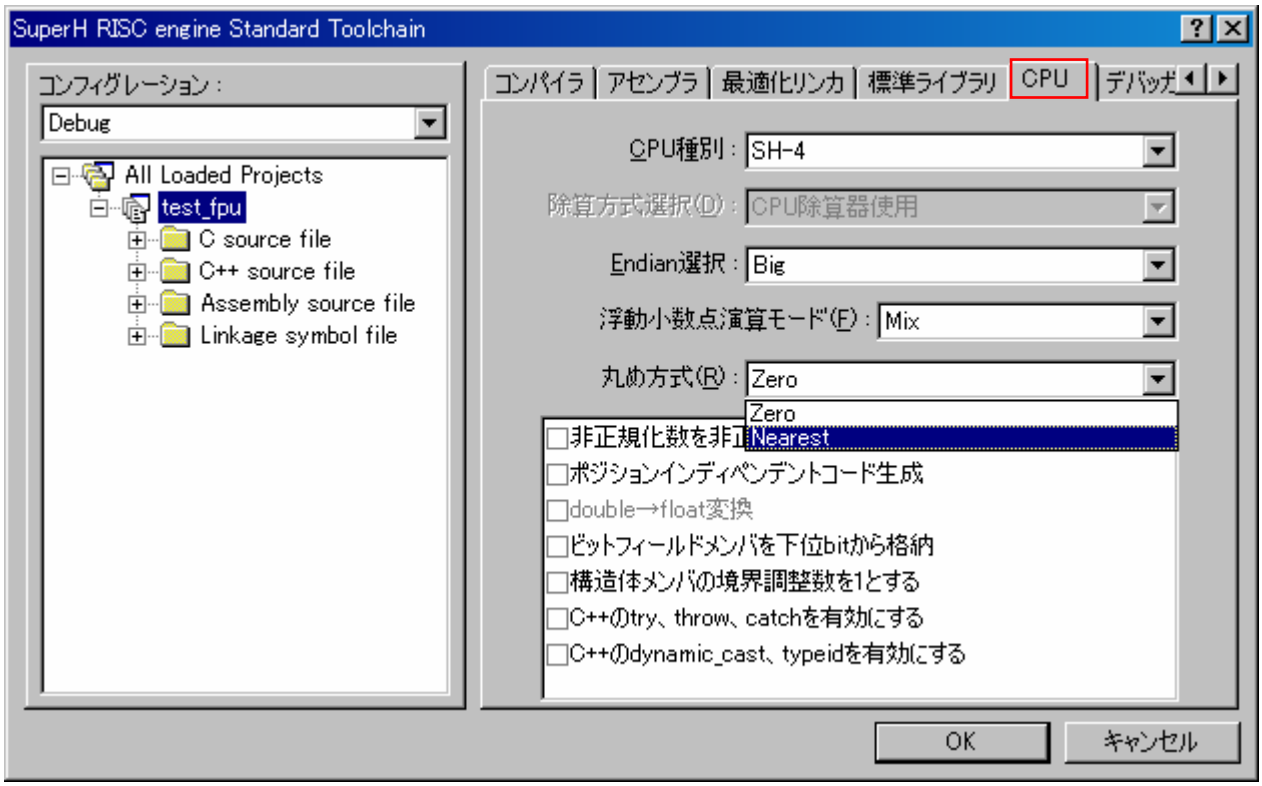


図 2-3

2.4 非正規化数の扱い(DENormalize={OFF|ON}) ＜SH-4、SH-4A＞

SH-4、SH-4Aでは非正規化数の扱い方を「非正規化数として扱う」もしくは「0 として扱う」から選択することができます。非正規化数の扱い方を設定する際には、FPSCRのDNビットの設定とコンパイラのdenormalizeオプションの設定を同じとする必要があります。コンパイラのdenormalizeオプションの設定は、図 2-4の ”非正規化数を非正規化数として扱う” のチェックボックスで設定できます。

- チェックボックスを選択しなかった場合(denormalize=off)[デフォルト]
  非正規化数を 0 として扱います。
- チェックボックスを選択した場合(denormalize=on)
  非正規化定数を非正規化数として扱います。

コンパイラは FPSCR の DN ビットの値を変更するコードは生成しませんので、DN ビットの設定はユーザプログラムにて明示的に行う必要があります。CPU の初期状態では FPSCR の DN ビットの値は「1(=非正規化数を 0 として扱う)」となっています。非正規化数を非正規化数として扱う場合は、DN ビットを明示的に「0(=非正規化数を非正規化数として扱う)」に設定してください。FPSCR は組み込み関数 set_fpscr で設定できます。

例:

ソースコード float ff = 1.0e-38f; denormalize=off 指定時(デフォルト)のアセンブリ展開コード _ff: .DATA.L H'00000000	denormalize=on 指定時のアセンブリ展開コード _ff: .DATA.L H'006CE3EE

[HEW でのオプション設定方法]

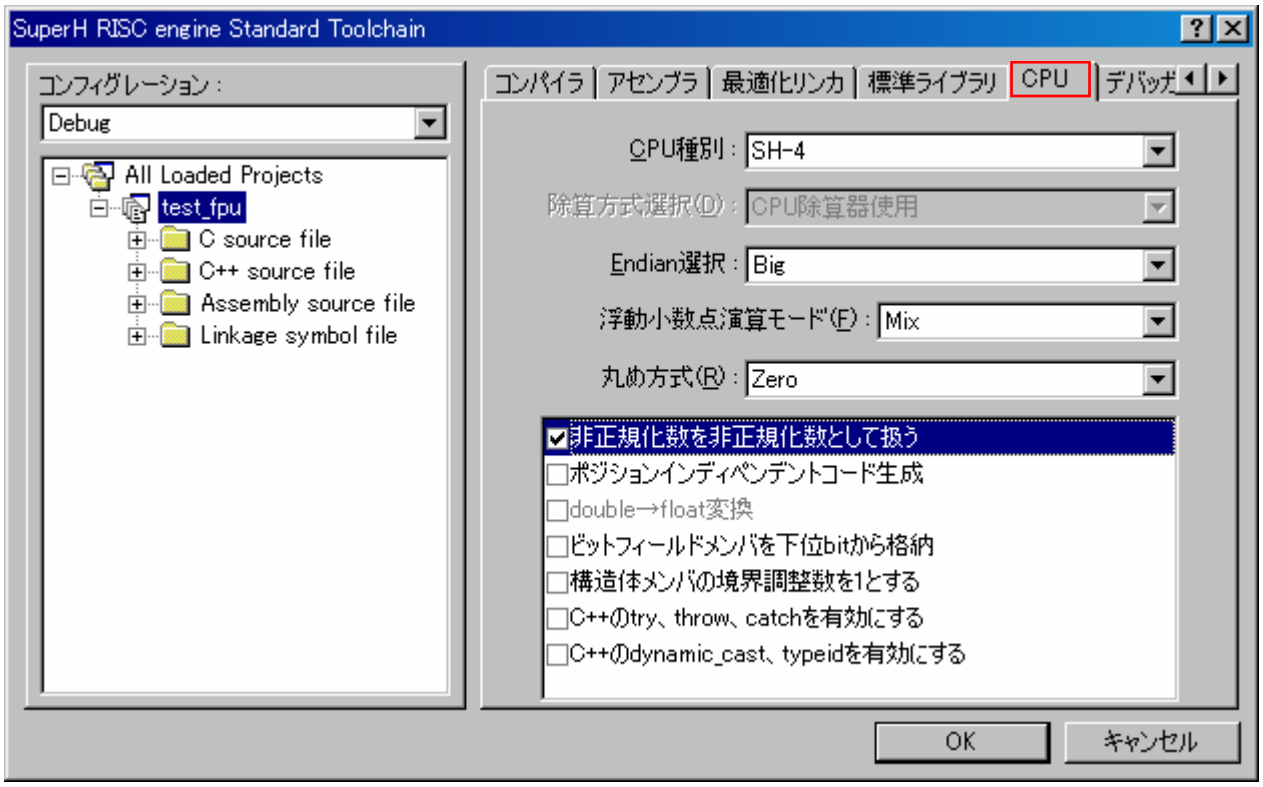


図 2-4

2.5 浮動小数点除算の乗算化(APproxdiv)

浮動小数点定数による除算を、その定数の逆数による乗算に置き換えるオプションです。除算を乗算に置き換えることにより、演算速度の向上が期待できます。図 2-5の "浮動小数点定数除算の乗算化" のチェックボックスを選択することでオプションapproxdivの指定が行えます。

このオプションは FPU の有無に関わらず指定できるオプションです。

[注意]

この最適化により、浮動小数点演算の誤差が変わる可能性があります。ご注意ください。

例: cpu=sh4 の場合

ソースコード

```
float x;

void f (float y)
{
    x=y/3.0f;
}
```

approxdiv未指定時(デフォルト)のアセンブリ展開コード

```
_f:
    MOVA        L11,R0
    MOV.L       L11+4,R2    ; _x
    FMOV.S      @R0,FR8
    FDIV        FR8,FR4     ;FR8=H'40400000
                            ;    (3.0)
    RTS
    FMOV.S      FR4,@R2     ; x
L11:
    .DATA.L     H'40400000
    .DATA.L     _x
```

approxdiv指定時のアセンブリ展開コード

```
_f:
    MOVA        L11,R0
    MOV.L       L11+4,R2    ; _x
    FMOV.S      @R0,FR8
    FMUL        FR8,FR4     ;FR8=H'3EAAAAAA
                            ;    (0.3333333…)
    RTS
    FMOV.S      FR4,@R2     ; x
L11:
    .DATA.L     H'3EAAAAAA
    .DATA.L     _x
```

3.0 による除算を逆数の乗算に置き換えます。

[HEW でのオプション設定方法]

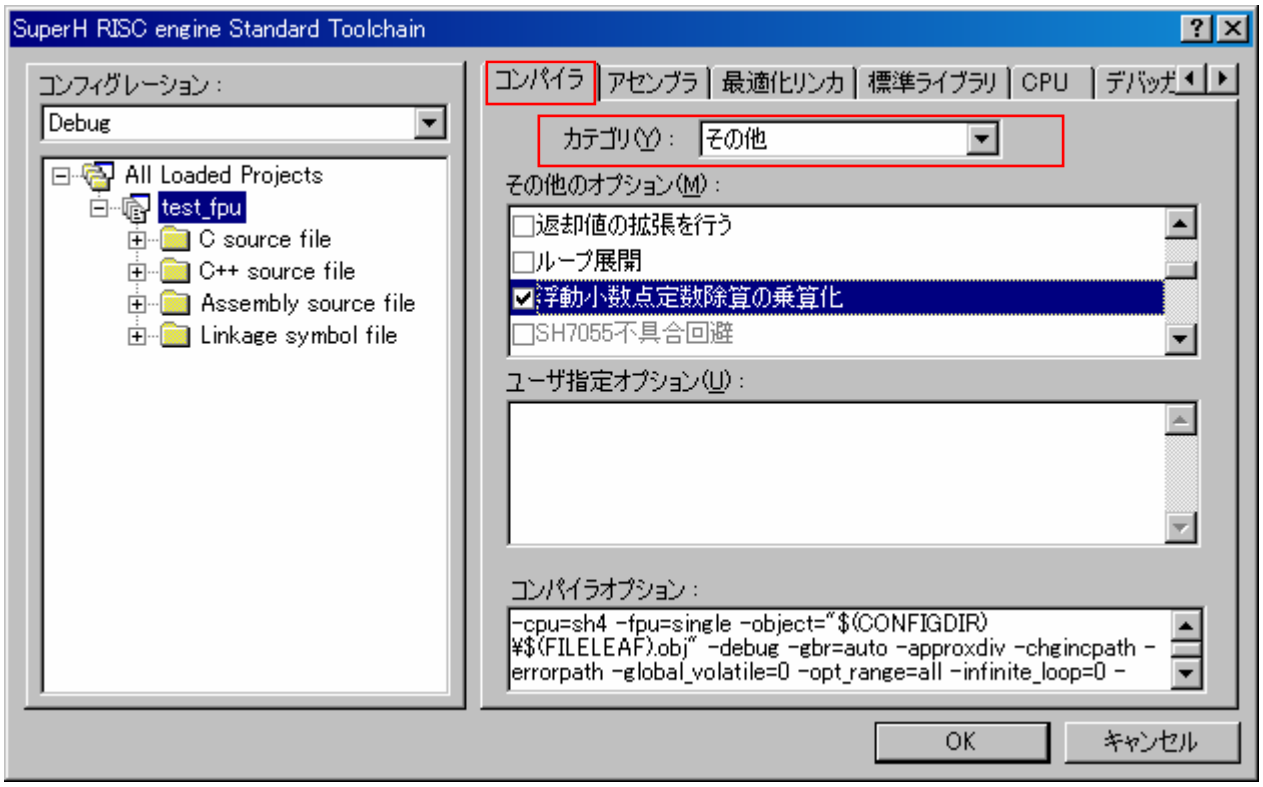


図 2-5

2.6 浮動小数点除算変換(FDIv)

整数除算を浮動小数点除算に置き換えるオプションです。実行時ルーチンで処理される整数除算をFPUの除算命令に置き換えることにより、演算速度の向上が期待できます。図 2-6の “整数除算の浮動小数点除算置き換え” のチェックボックスを選択することで、オプションfdivの指定が行えます。

[補足]

cpu=sh2afpu|sh4|sh4a かつ fpu=double を指定した場合は除数、被除数が共に 4byte 以内の時に、
それ以外の場合は除数、被除数が共に 2byte 以内の時に変換を行います。

例: cpu=sh4 fpu=double の場合

ソースコード

```
int x;

func(int a, int b)
{
    x = a/b;
}
```

fdiv未指定時(デフォルト)のアセンブリ展開コード

```
_func:
    STS.L       PR,@-R15
    MOV.L       L11+2,R2    ; __divls
    MOV         R4,R1
    MOV.L       L11+6,R6    ; _x
    JSR         @R2
    MOV         R5,R0
    LDS.L       @R15+,PR
    RTS
    MOV.L       R0,@R6      ; x
L11:
    .RES.W      1
    .DATA.L     __divls
    .DATA.L     _x
```

fdiv指定時のアセンブリ展開コード

```
_func:
    LDS         R4,FPUL
    MOV.L       L11,R6      ; _x
    FLOAT       FPUL,DR6
    LDS         R5,FPUL
    FLOAT       FPUL,DR8
    FDIV        DR8,DR6
    FTRC        DR6,FPUL
    STS         FPUL,R2
    RTS
    MOV.L       R2,@R6      ; x
L11:
    .DATA.L     _x
```

[HEW でのオプション設定方法]

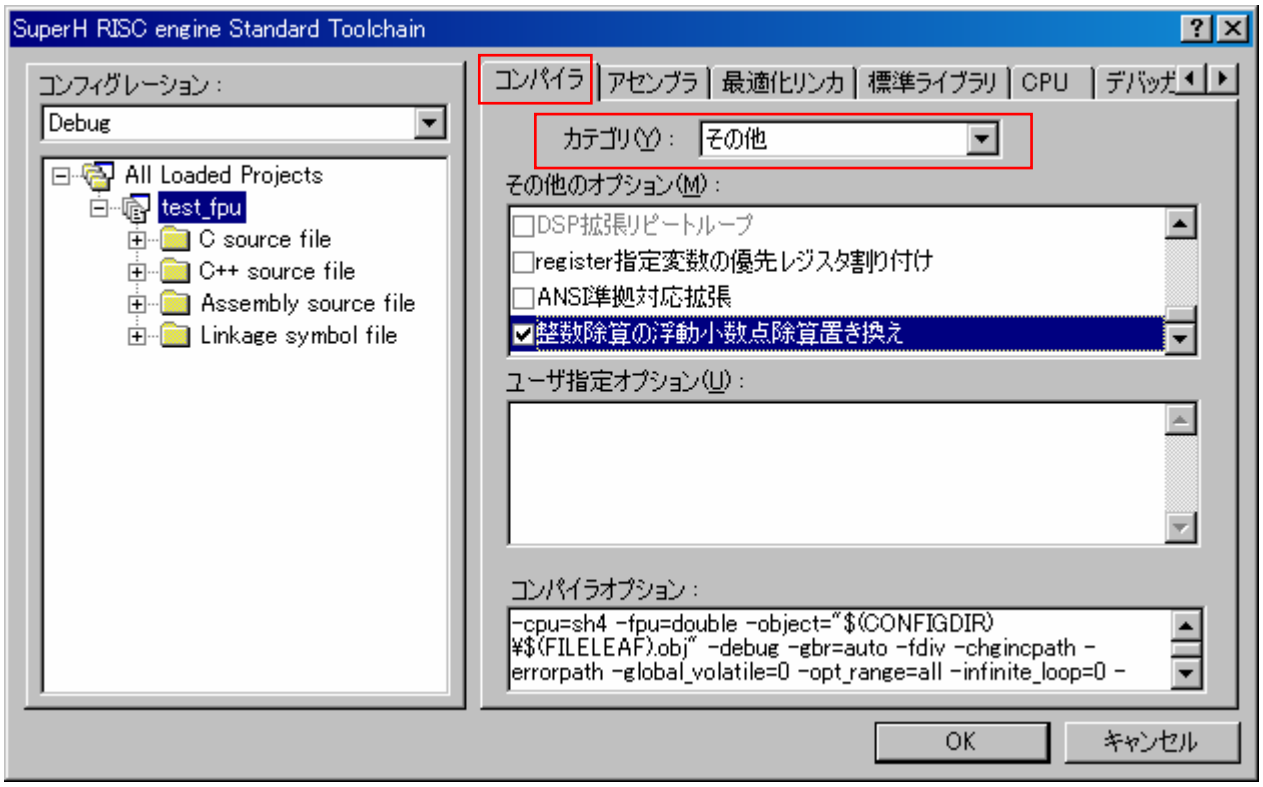


図 2-6

2.7 FPSCR レジスタ精度モード切り替え(FPScr={Aggressive|Safe}) ＜SH2A-FPU、SH-4、SH-4A＞

FPSCR の精度モード(PR ビット)を関数呼び出し前後で保証するか否かを指定するオプションです。fpscr=aggressivee(デフォルト)では、関数呼び出しの前後で PR ビットの値を保証しません。
fpscr=safe では、関数呼び出しの後は常に単精度となるよう、呼び出された関数側で PR ビットの値を保証します。

fpscr=aggressivee(デフォルト)の場合、関数呼び出しから戻った時の PR ビットの値は不明です。その為、関数呼び出し後に浮動小数点演算がある時は、FPSCR の値を再設定するコードが常に生成されます。しかし、fpscr=safe の場合は、関数呼び出しから戻った時は PR ビットが単精度と保証されていますので必要な場合のみ FPSCR が設定されます。その為、fpscr=safe の方が効率の良いコードが生成される傾向にあります。

fpscr=safe を指定するには、図 2-7の “FPSCRレジスタの切り替え” のチェックボックスを選択してください。

[注意]

本オプションは関数のインタフェースを変更するため、全ファイルに対して同時に変更する必要があります。
旧バージョンコンパイラで作成したライブラリをリンクする時は特に注意してください。

例: cpu=sh4 の場合

ソースコード

```
extern void sub(void);
extern float f1, f2;

func()
{
    sub();
    f1 =1.0f;
}
```

fpscr=aggressive指定時(デフォルト)のアセンブリ展開コード

```
_func:
        STS.L       PR,@-R15
        MOV.L       L11,R1      ; _sub
        JSR         @R1
```

fpscr=safe指定時のアセンブリ展開コード

```
_func:
        STS.L       PR,@-R15
        MOV.L       L11,R1      ; _sub
        JSR         @R1
```

```
        NOP
        STS         FPSCR,R4
        MOV.L       L11+4,R6    ; H'FFE7FFFF
        MOVA        L11+8,R0
        MOV.L       L11+12,R5   ; _f1
        AND         R6,R4
        LDS         R4,FPSCR
        FMOV.S      @R0,FR8
        LDS.L       @R15+,PR
        RTS
        FMOV.S      FR8,@R5     ; f1
L11:
        .DATA.L     _sub
        .DATA.L     H'FFE7FFFF
        .DATA.L     H'3F800000
        .DATA.L     _f1
```

```
        NOP
        MOVA        L11+4,R0
        MOV.L       L11+8,R4    ; _f1
        FMOV.S      @R0,FR8
        LDS.L       @R15+,PR
        RTS
        FMOV.S      FR8,@R4     ; f1
L11:
        .DATA.L     _sub
        .DATA.L     H'3F800000
        .DATA.L     _f1
```

[HEW でのオプション設定方法]

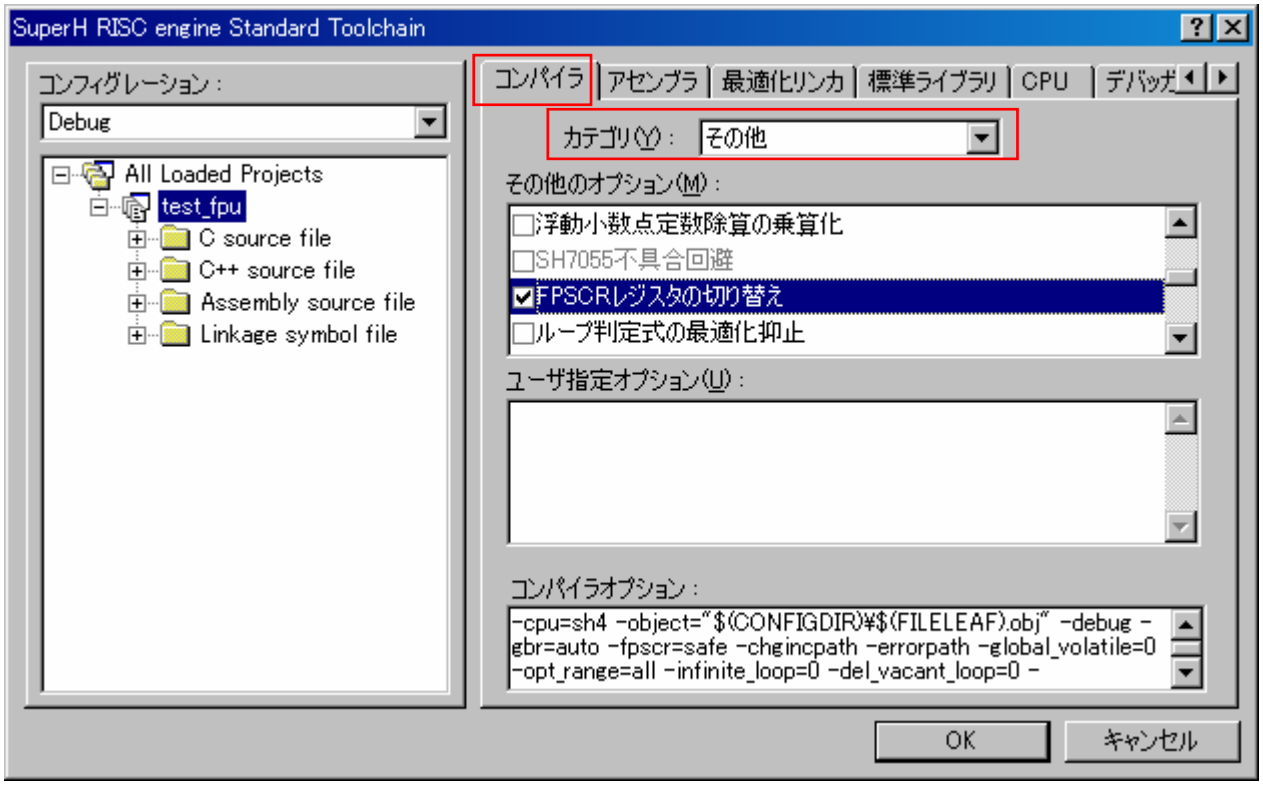


図 2-7

2.8 浮動小数点数-整数変換時の範囲チェック省略(SIMple_float_conv)

符号なし整数型と浮動小数点数型の間の型変換に対して、変換対象の値の範囲チェックを省略したコードを生成するためのオプションです。値の範囲チェックを省略するため、演算速度が向上します。ただし、型変換前の値が 0 以上 2147483647 以下の整数値もしくは 0.0 以上 2147483647.0 以下の浮動小数点数値では無い場合は、演算結果が不正となります。これら範囲外の値が入力される可能性がある時は本オプションを使用しないでください。

例: cpu=sh4 の場合

- float 型から unsigned int 型への変換時

ソースコード

```
unsigned long func(float f)
{
    return ((unsigned int)f);
}
```

simple_float_conv未指定時(デフォルト)のアセンブリ展開コード

```
_func:
    MOV         #79,R2  ; 0x0000004F
    SHLL8       R2      ;
    SHLL16      R2      ; 0x4F000000
    LDS         R2,FPUL
    FSTS        FPUL,FR8
    FCMP/GT     FR4,FR8
    BT          L12
    FADD        FR8,FR8  ;f≧0x4F000000の場合
    FSUB        FR8,FR4  ;変換前の値を
                         ;(f-0x4F800000)とする
L12:
    FTRC        FR4,FPUL ;float→int変換
    RTS
    STS         FPUL,R0
```

simple_float_conv指定時のアセンブリ展開コード

```
_func:
    FTRC        FR4,FPUL    ;float→int変換
    RTS
    STS         FPUL,R0
```

- unsigned int 型から float 型への変換時

ソースコード

```
float func(unsigned int ui)
{
    return ((float)ui);
}
```

simple_float_conv未指定時(デフォルト)のアセンブリ展開コード

```
_func:
    LDS         R4,FPUL
    CMP/PZ      R4
    BT/S        L12
    FLOAT       FPUL,FR0 ;int → float変換
    MOVA        L13+2,R
    FMOV.S      @R0,FR9  ;u≧0x80000000uの場合、
    FADD        FR9,FR0  ;変換後の値に
                         ;0x4F800000を加算する。
L12:
    RTS
    NOP
L13:
    RES.W       1
    DATA.L      H'4F800000
```

simple_float_conv指定時のアセンブリ展開コード

```
_func:
    LDS         R4,FPUL
    RTS
    FLOAT       FPUL,FR0 ;int→float変換
```

[HEW でのオプション設定方法]

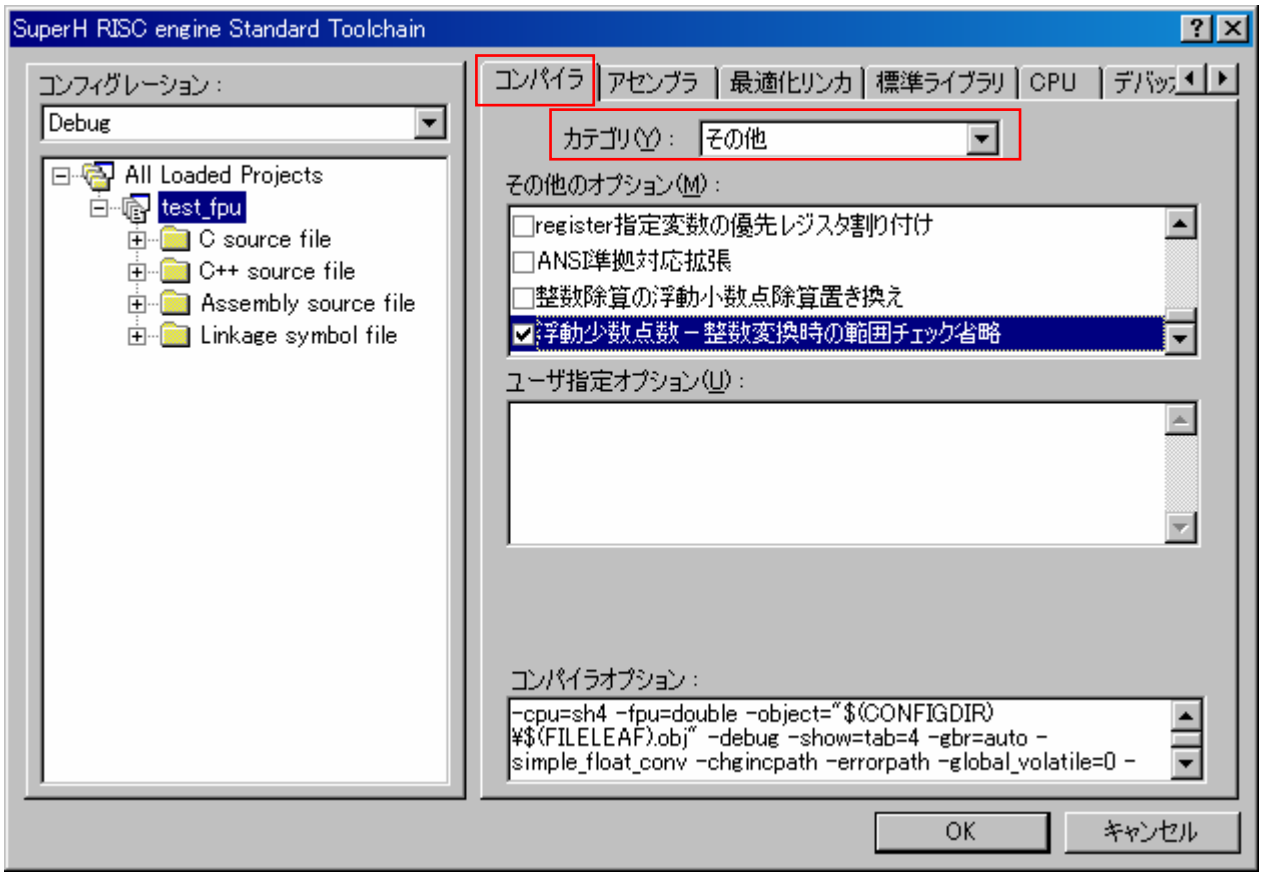


図 2-8

2.9 浮動小数点レジスタ退避/回復抑止(IFUnc、#pragma ifunc)

浮動小数点レジスタの退避/回復処理を抑止するためのオプションおよび#pragma です。本機能は割込み関数指定（#pragma interrupt ）と組み合わせて使用します。

割込み関数の中に関数呼び出しがある場合は、呼び出した関数内でどのレジスタが使用されるか判断できないため、浮動小数点レジスタを含めた全レジスタを退避/回復します。しかし、呼び出した関数内で浮動小数点演算を使用していない時は､浮動小数点レジスタの退避/回復は省略できます｡オプション ifunc もしくは #pragma ifunc を指定することで、浮動小数レジスタの退避回復を抑止させることができます。

オプション ifunc を用いると、ソースファイル内すべての関数に対する指定となります。#pragma ifunc は特定の関数に対しての指定となります。ifunc および #pragma ifunc は、割り込み関数と割り込み関数から呼び出される関数の両方に指定する必要があります。割り込み関数から呼び出される関数が浮動小数点命令を必要とする場合はコンパイルエラー

C2843 (E) Illegal floating type used in function

が通知されます。また、割り込み関数から ifunc、 #pragma ifunc の指定がない関数が呼び出された場合は、ウォーニング

C1029 (W) Function with ifunc calls "関数名" without ifunc

を通知します（ウォーニングが通知されても浮動小数点レジスタ退避/回復抑止は有効のままとなります）。

例: cpu=sh4 の場合

ソースコード(#pragma ifunc 指定なし)

```
#pragma interrupt (func)

void sub()
{
}
void func()
{
   sub();
}
```

ソースコード(#pragma ifunc 指定あり)

```
#pragma interrupt (func)
#pragma ifunc (sub,func)
void sub()
{
}
void func()
{
   sub();
}
```

アセンブリ展開コード

```
_sub:
        RTS
        NOP
_func:
        MOV.L       R0,@-R15
        MOV.L       R1,@-R15
        MOV.L       R2,@-R15
        MOV.L       R3,@-R15
        MOV.L       R4,@-R15
        MOV.L       R5,@-R15
        MOV.L       R6,@-R15
        MOV.L       R7,@-R15
        STS.L       PR,@-R15
        FMOV.S      FR0,@-R15
        FMOV.S      FR1,@-R15
        FMOV.S       FR2,@-R15
        FMOV.S      FR3,@-R15
        FMOV.S      FR4,@-R15
        FMOV.S      FR5,@-R15
        FMOV.S      FR6,@-R15
        FMOV.S      FR7,@-R15
        FMOV.S      FR8,@-R15
        FMOV.S      FR9,@-R15
        FMOV.S      FR10,@-R15
        FMOV.S      FR11,@-R15
        STS.L       FPUL,@-R15
        STS.L       FPSCR,@-R15
        STC         SSR,@-R15
        STC         SPC,@-R15
        BSR         _sub
        NOP
        LDC         @R15+,SPC
        LDC         @R15+,SSR
        LDS.L       @R15+,FPSCR
        LDS.L       @R15+,FPUL
        FMOV.S      @R15+,FR11
        FMOV.S      @R15+,FR10
        FMOV.S      @R15+,FR9
        FMOV.S      @R15+,FR8
        FMOV.S      @R15+,FR7
        FMOV.S      @R15+,FR6
        FMOV.S      @R15+,FR5
        FMOV.S      @R15+,FR4
        FMOV.S      @R15+,FR3
        FMOV.S      @R15+,FR2
        FMOV.S      @R15+,FR1
        FMOV.S      @R15+,FR0
        LDS.L       @R15+,PR
        MOV.L       @R15+,R7
        MOV.L       @R15+,R6
        MOV.L       @R15+,R5
        MOV.L       @R15+,R4
        MOV.L       @R15+,R3
        MOV.L       @R15+,R2
        MOV.L       @R15+,R1
        MOV.L       @R15+,R0
        RTE
```

アセンブリ展開コード

```
_sub:
        RTS
        NOP
_func:
        MOV.L       R0,@-R15
        MOV.L       R1,@-R15
        MOV.L       R2,@-R15
        MOV.L       R3,@-R15
        MOV.L       R4,@-R15
        MOV.L       R5,@-R15
        MOV.L       R6,@-R15
        MOV.L       R7,@-R15
        STS.L       PR,@-R15
        STC         SSR,@-R15
        STC         SPC,@-R15
        BSR         _sub
        NOP
        LDC         @R15+,SPC
        LDC         @R15+,SSR
        LDS.L       @R15+,PR
        MOV.L       @R15+,R7
        MOV.L       @R15+,R6
        MOV.L       @R15+,R5
        MOV.L       @R15+,R4
        MOV.L       @R15+,R3
        MOV.L       @R15+,R2
        MOV.L       @R15+,R1
        MOV.L       @R15+,R0
        RTE
        NOP
```

(1) #pragma ifunc の指定

書式: #pragma ifunc [(]<関数名>[)]

説明: <関数名>で指定した関数での浮動小数点レジスタの退避/回復を抑止します。

備考:

#pragma ifunc 指定は、関数の宣言前に行ってください。

#pragma ifunc 指定された関数内で浮動小数点を使用した場合はコンパイルエラーとなります。

(2)オプション ifunc の指定
ファイル単位で指定します。浮動小数点命令が生成されるソースプログラムに指定するとコンパイルエラーとなります。

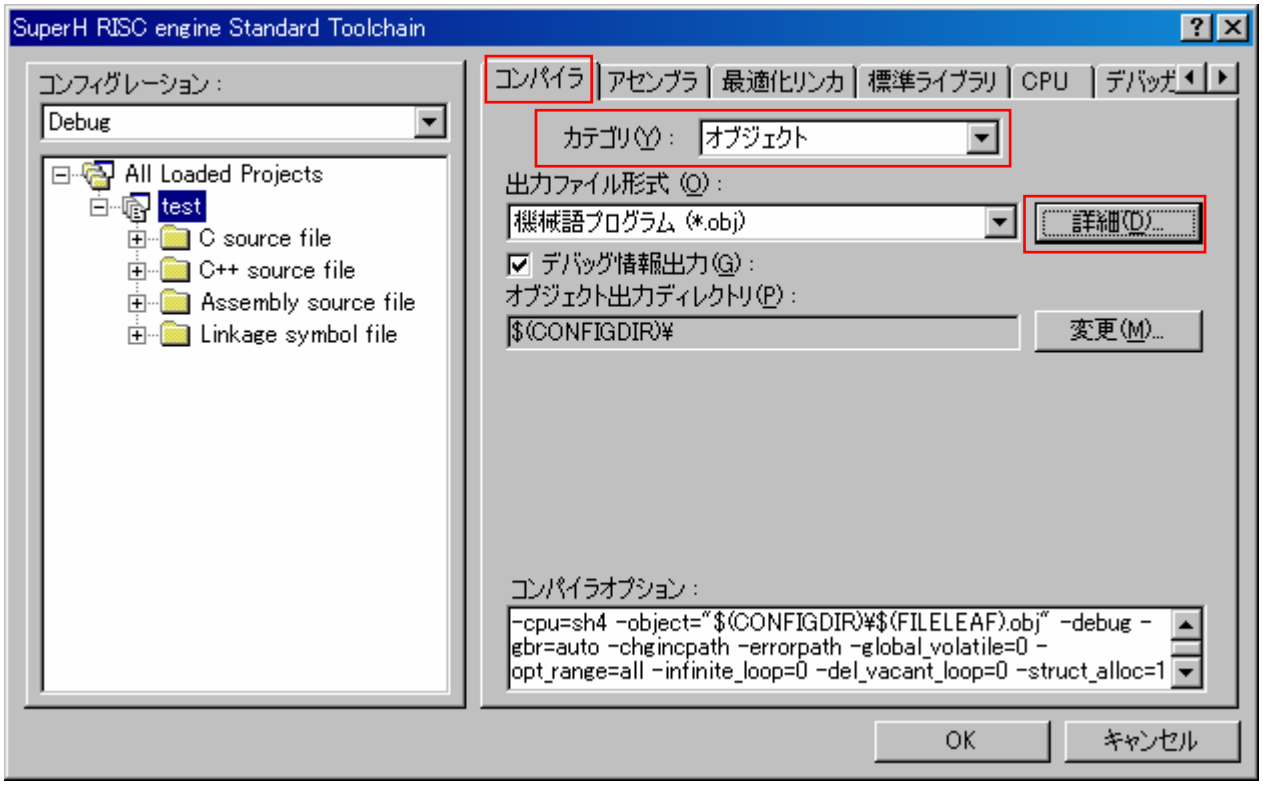


図 2-9

“詳細”ダブを選択 → “浮動小数点レジスタ退避・回復抑止”のチェックボックスを選択

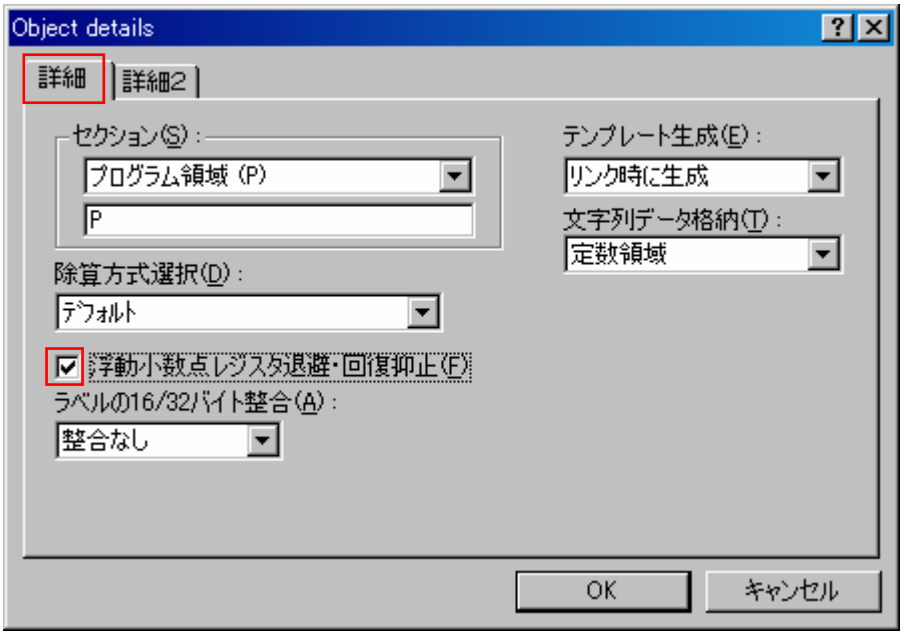


図 2-10

3. 効率のよいプログラミング技法

3.1 浮動小数点命令の活用

■ポイント

浮動小数点命令 FABS (SH-2E、SH2A-FPU、SH-4、SH-4A)、FSQRT (SH2A-FPU、SH-4、SH-4A)を活用するためには, 単精度の場合は、インクルードファイル <mathf.h> をインクルードし、単精度の浮動小数点関数 fabsf, sqrtf を呼び出します。倍精度の場合は、インクルードファイル <math.h> をインクルードし、倍精度の浮動小数点関数 fabs、sqrt を呼び出します。

■説明

単精度の浮動小数点命令 FABS (SH-2E、SH2A-FPU、SH-4、SH-4A)、FSQRT (SH2A-FPU、SH-4、SH-4A) を活用するためには、以下のようにしてください。

(a) <mathf.h> をインクルードする。
(b) fabsf 関数 (FABS)、sqrtf 関数 (FSQRT) を呼び出す。

倍精度の浮動小数点命令 FABS (SH2A-FPU、SH-4、SH-4A)、FSQRT (SH2A-FPU、SH-4、SH-4A) を活用するためには、以下のようにしてください。

(a) <math.h> をインクルードする。
(b)fabs 関数 (FABS)、sqrt 関数 (FSQRT) を呼び出す。

■使用例

改善前の例では、<mathf.h> をインクルードしていないため、コンパイラは標準の関数であると認識せず、ライブラリから fabsf 関数を呼び出します。<mathf.h> をインクルードすれば、コンパイラが FABS 命令に対応する関数であることを認識でき、FABS 命令を直接生成します。

[注意]
ヘッダ <mathf.h> は ANSI 標準の C ライブラリ機能ではありません。

改善前ソースコード

```
float fabsf(float);

float f(float x, float y){
    return fabsf(x)+fabsf(y);
}
```

改善前アセンブリコード

```
_f:
          STS.L       PR,@-R15
          FMOV.S      FR14,@-R15
          FMOV.S      FR15,@-R15
          MOV.L       L11+2,R1    ; _fabsf
          JSR         @R1
          FMOV.S      FR5,FR15
          FMOV.S      FR0,FR14
          MOV.L       L11+2,R4    ; _fabsf
          JSR         @R4
          FMOV.S      FR15,FR4
          FADD        FR0,FR14
          FMOV.S      FR14,FR0
          FMOV.S      @R15+,FR15
          FMOV.S      @R15+,FR14
          LDS.L       @R15+,PR
          RTS
          NOP
L11:
          .RES.W      1
          .DATA.L     _fabsf
```

改善後ソースコード

```
#include <mathf.h>

float f(float x, float y){
    return fabsf(x)+fabsf(y);
}
```

改善後アセンブリコード

```
_f:
          FABS        FR4
          FABS        FR5
          FADD        FR5,FR4
          RTS
          FMOV.S      FR4,FR0
```

4. よくある問い合わせ

4.1 浮動小数点演算結果

■質問

浮動小数点演算で演算結果が期待値とならない。

■回答 1

丸め誤差のため期待値となっていない可能性があります。
例えば、リスト 4-1の変数 a は double型のため、内部では浮動小数点数で表現されます。 round=zeroオプションを選択している場合では、この変数a の内部表現は[0x4023FFFFFFFFFFFF]となります。この値は 10 ではなく 10 に限りなく近い 9.9999・・・・ という近似値です。その為、リスト 4-1の変数bには小数点以下が切り捨てられた 9 が代入されます。

```
(サンプルプログラム)
double a = 0.1 * 100;
int b;

void func(void)
{
    b = (int) a;
}
```

リスト 4-1

本現象は浮動小数点数表現の誤差のため発生します。したがって根本的解決方法はありません。以下のように誤差を考慮したコーディングをしてください。

```
(誤差を考慮しないコーディング例)
float f;

if ( f == 0.1f ) {
        .
        .
        .
}
```

```
(誤差を考慮したコーディング例)
const float s = 1.0e-10f;
float f;

if ((0.1f-s) <= f && f <= (0.1f+s)) {
        .
        .
        .
}
```

リスト 4-2

■回答 2

SH2A-FPU、SH-4、SH-4A の場合、オプションと FPCSR の値との対応が取れていない可能性があります。以下を確認してください。

1. コンパイラの fpu オプションと、FPCSR の PR ビット値の対応を確認してください。
2. コンパイラの denormalize オプションと、FPCSR の DN ビット値の対応を確認してください（SH-4、SH-4A のみ）。
3. コンパイラの round オプションと、FPCSR の RM ビット値の対応を確認してください（SH-4、SH-4A のみ）。

[浮動小数点演算モード]

fpu オプションは、fpu=single, double の 2 種類が選択可能です。
"fpu=single" を選択した場合、全ての浮動小数点データを単精度浮動小数点データとして扱います。
"fpu=double" を選択した場合、全ての浮動小数点データを倍精度浮動小数点データとして扱います。
fpu オプションを使用しない場合、ソースコードの宣言型通りに浮動小数点データを扱います。

fpu オプションを使用しない場合、コンパイラは、浮動小数点の演算時にその精度に応じて PR ビットを変更するコードを生成しますが、"fpu= single" または "fpu=double" を選択した場合には、コンパイラは PR ビットにアクセスするコードを一切生成しません。

一方、FPSCR レジスタの PR ビットは、リセット時に 0 に初期化されます。
そのため、fpu オプションを使用しない場合、または "fpu=single" を選択した場合には、PR ビットを意識しなくても正しく動作しますが、"fpu=double" を選択した場合には、FPU 演算前に、PR ビットを 1 に変更する必要があります。

[非正規化数の扱い]

"denormalize=on" オプションを選択している場合には、FPSCR の DN ビットが 0 に設定されていることを確認してください。

"denormalize=off" オプションを選択している場合には、FPSCR の DN ビットが 1 に設定されていることを確認してください。

[丸め方式]

"round=zero" オプションを選択している場合には、FPSCR の RM ビットが 01 に設定されていることを確認してください。

"round=nearest" オプションを選択している場合には、FPSCR の RM ビットが 00 に設定されていることを確認してください。

4.2 浮動小数点数のウォッチウィンドウの値

■質問

浮動小数点数の値について、ウォッチウィンドウで表示されている値と実際の値が違っています。

■回答

ウォッチウィンドウでは浮動小数点数の値が近似値で表示される場合があります。

例えば、内部表現 [0x4023FFFFFFFFFFFF]となっている浮動小数点数の実際の値は 9.9999・・・・ですが、ウォッチウィンドウ(図 4-1)で表示される値は 10 となります。

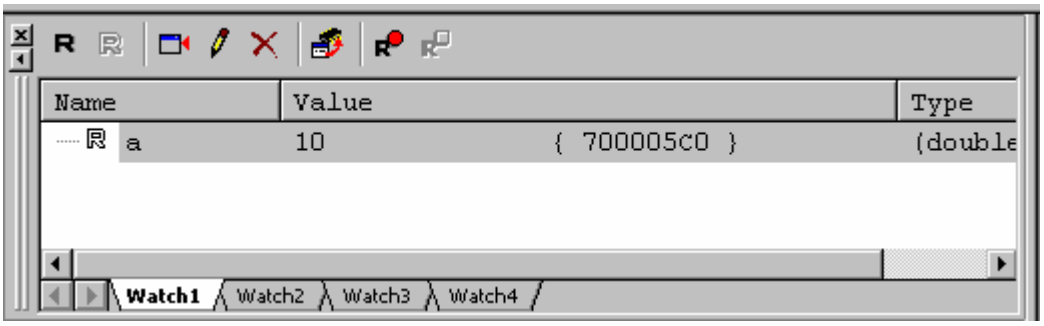


図 4-1

実際の値は、メモリウィンドウ(図 4-2)もしくは、レジスタウィンドウ(図 4-3)で内部表現を確認することができます。

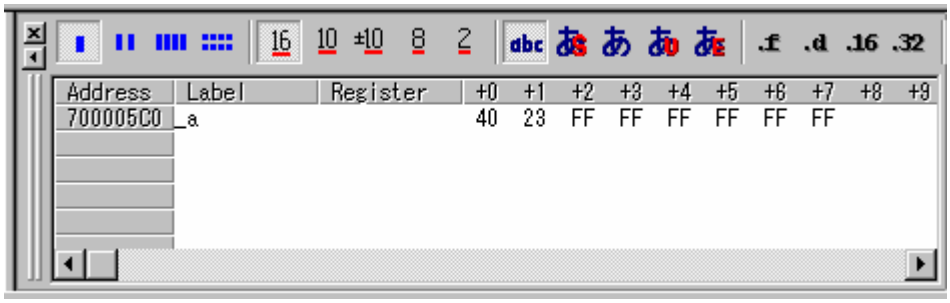


図 4-2

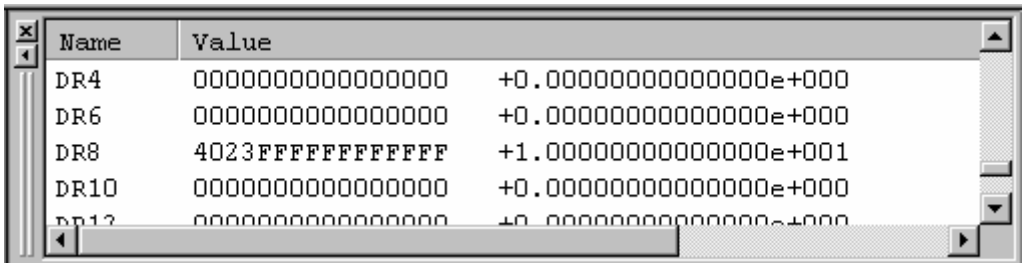


図 4-3

ホームページとサポート窓口<website and support,ws>

ルネサステクノロジホームページ

http://japan.renesas.com/

お問合せ先

http://japan.renesas.com/inquiry

csc@renesas.com

改訂記録<revision history,rh>

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2007.6.1	—	初版発行

安全設計に関するお願い

1. 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故、火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

1. 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
2. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
3. 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
4. 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
5. 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
6. 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
7. 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
8. 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。

© 2007. Renesas Technology Corp., All rights reserved.