RX 開発環境移行ガイド

H8 から RX への移行（コンパイラ編）
（High-performance Embedded Workshop, H8C→CS+,CC-RX）

RJJ06J0077-0200
Rev.2.00
2016.11.30

要旨

本ドキュメントでは、ルネサス製 H8SX、H8S、H8 ファミリ（以下、H8 ファミリと略します）用コンパイラで作成したCプログラムをルネサス製 RX ファミリ用コンパイラへ移植する際のソフトウェア移行方法を説明します。

RX ファミリ用コンパイラでは、H8 ファミリから RX ファミリへの移行を考慮し、オプションと言語仕様の差異を吸収するもしくは診断する機能をサポートしています。これにより、組み込みソフトウェアのアプリケーション部分をスムーズに移行することができます。

H8 ファミリから RX ファミリへの移行時に活用下さい。

目次

1.　オプション

H8 ファミリ用コンパイラと RX ファミリ用コンパイラはデフォルトオプションで異なる仕様が存在します。H8 から RX への移行に際し、対応が必要になる可能性の高いオプションについて説明します。

表 1-1　オプション一覧

No	機能	H8 オプション	RX オプション	参照
1	char 型の符号指定	－	signed_char	1.1
2	ビットフィールドメンバの符号指定	－	signed_bitfield	1.2
3	ビットフィールドメンバの割り付け指定	bit_order	bit_order	1.3
4	エンディアン指定	－	endian	1.4
5	double 型のサイズ指定	double=float	dbl_size	1.5
6	int 型のサイズ違いへの対応	－	int_to_short	1.6

1.1　　char 型の符号指定

H8 ファミリ用コンパイラでは符号指定のない char 型は、符号ありの signed char 型として扱います。対して RX ファミリ用コンパイラではデフォルトでは符号なしの unsigned char 型として扱います。char 型が符号ありの signed char 型であることを前提に作成した H8 のプログラムを RX に移行するには、”signed_char”オプションを指定します。

【書式】

signed_char　　：デフォルトは unsigned_char

unsigned_char

[CS+でのオプション設定方法]

CC-RX（ビルド・ツール）プロパティの"共通オプション"タブ内で次のように設定します。

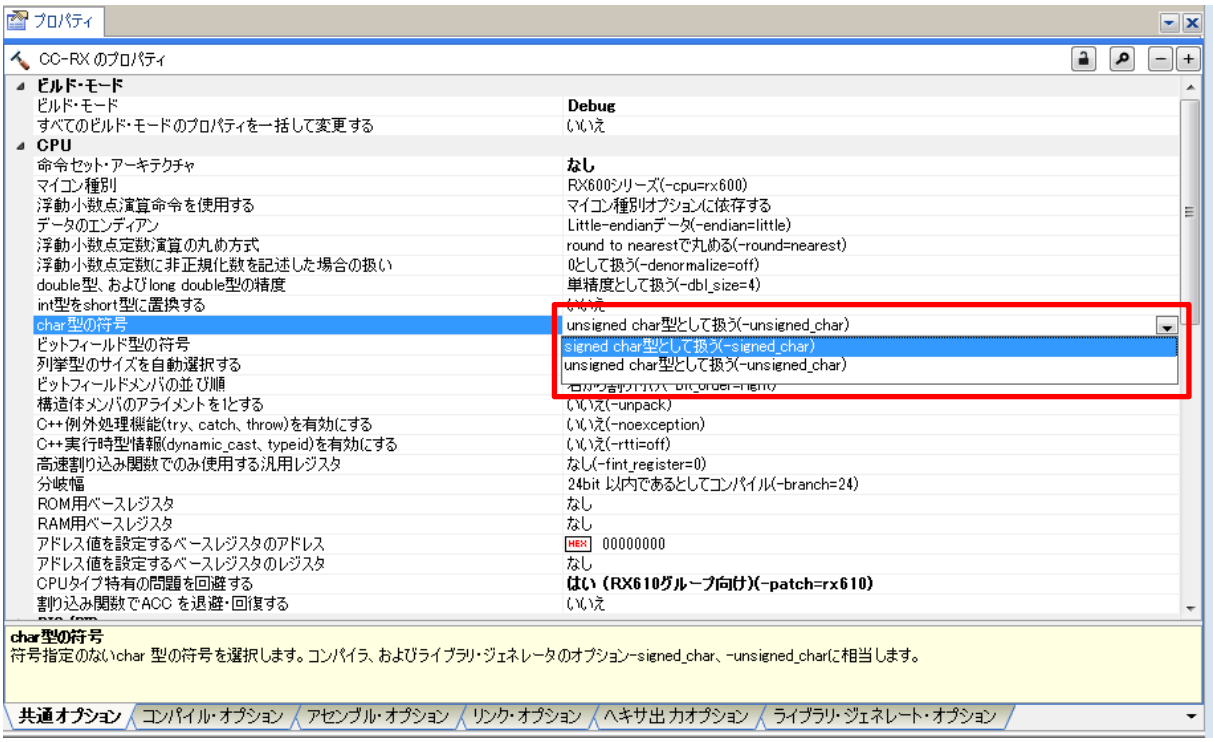

図 1-1

1.2　ビットフィールドメンバの符号指定

H8 ファミリ用コンパイラでは符号指定のないビットフィールドメンバは、符号ありの型として扱います。対して RX ファミリ用コンパイラではデフォルトでは符号なしの型として扱います。

符号指定のないビットフィールドメンバが、符号ありの型であることを前提に作成した H8 のプログラムを RX に移行するには、”signed_bitfield”オプションを指定します。

【書式】

signed_bitfield　　:デフォルトは unsigned_bitfield

unsigned_bitfield

[CS+でのオプション設定方法]

CC-RX（ビルド・ツール）プロパティの"共通オプション"タブ内で次のように設定します。

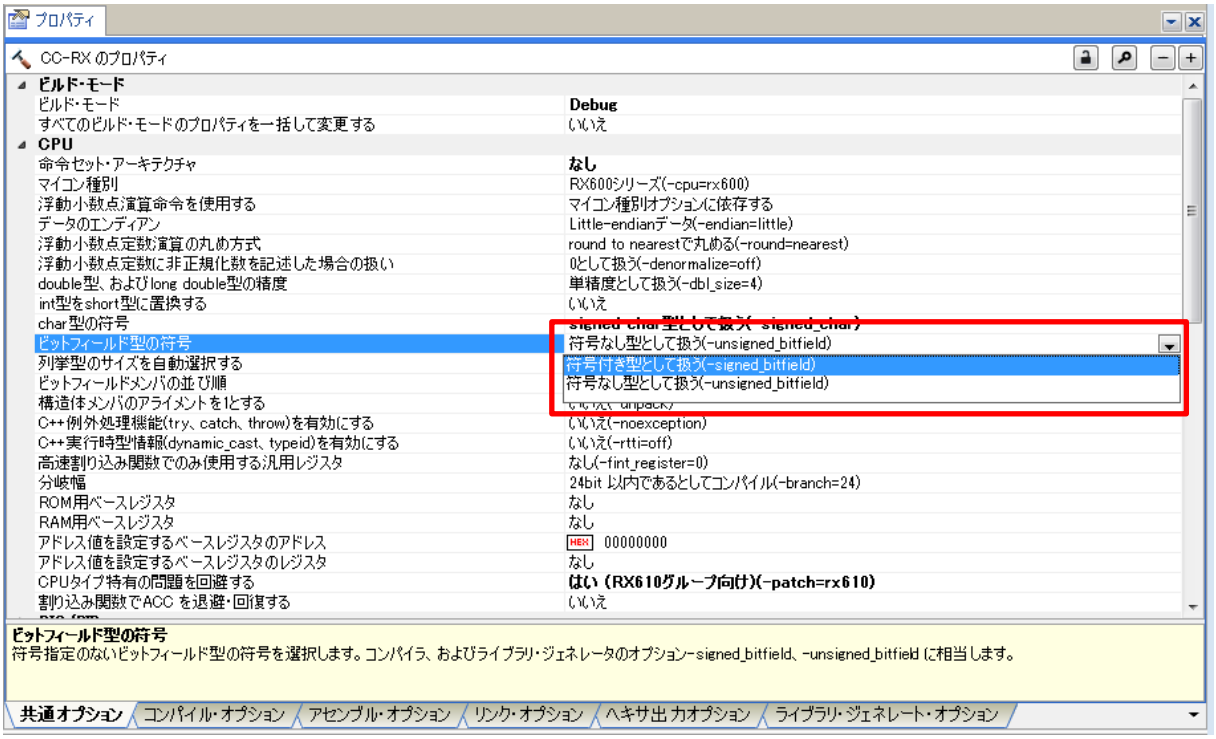

図 1-2

1.3　ビットフィールドメンバの割り付け指定

H8 ファミリ用コンパイラではビットフィールドメンバを上位ビットから割り付けます。対して RX ファミリ用コンパイラではデフォルトでは下位ビットから割り付けます。ビットフィールドメンバを上位から割り付けることを前提に作成した H8 のプログラムを RX に移行するには、”bit_order=left”オプションを指定します。

【書式】

bit_order={left|right}　　:デフォルトは right

[CS+でのオプション設定方法]

CC-RX（ビルド・ツール）プロパティの"共通オプション"タブ内で次のように設定します。

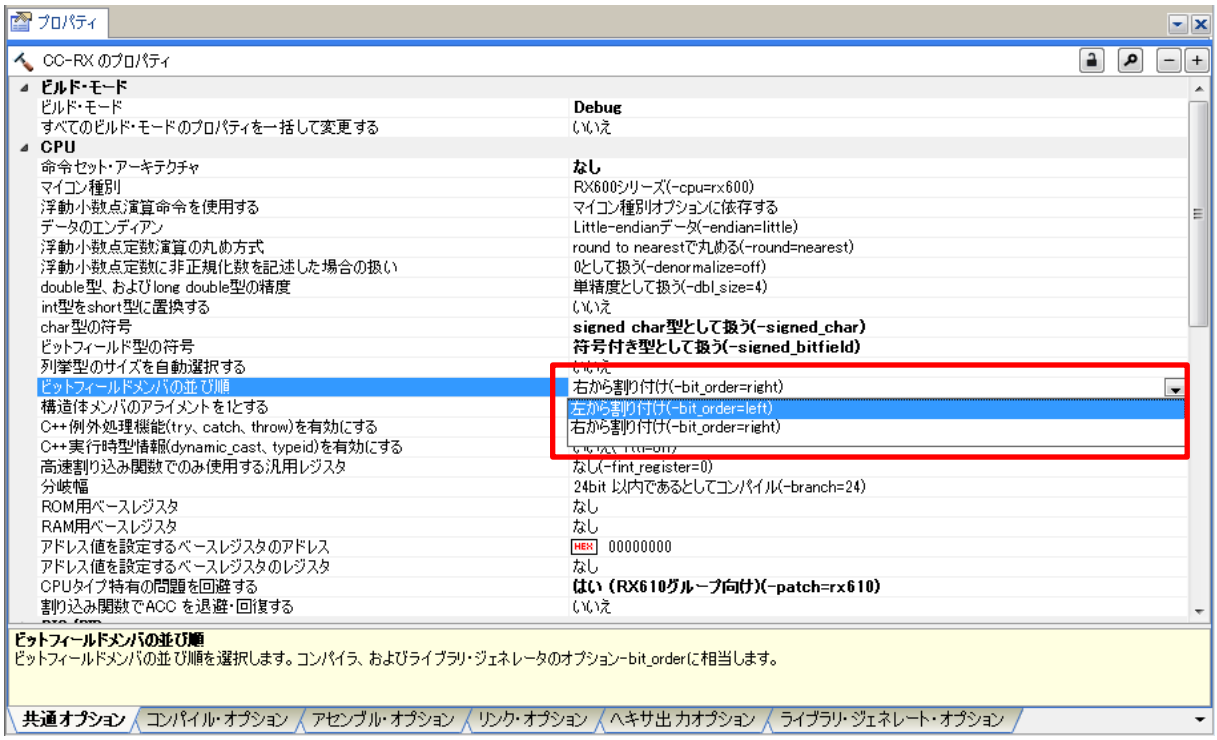

図 1-3

1.4　エンディアン指定

H8 ファミリ用コンパイラではデータのバイト並びが big endian になります。対して RX ファミリ用コンパイラではデフォルトでは little endian になります。

データのバイト並びが big endian であることを前提に作成した H8 のプログラムを RX に移行するには、”endian=big”オプションを指定します。

【書式】

endian={big|little}　　:デフォルトは little

[CS+でのオプション設定方法]

CC-RX（ビルド・ツール）プロパティの"共通オプション"タブ内で次のように設定します。

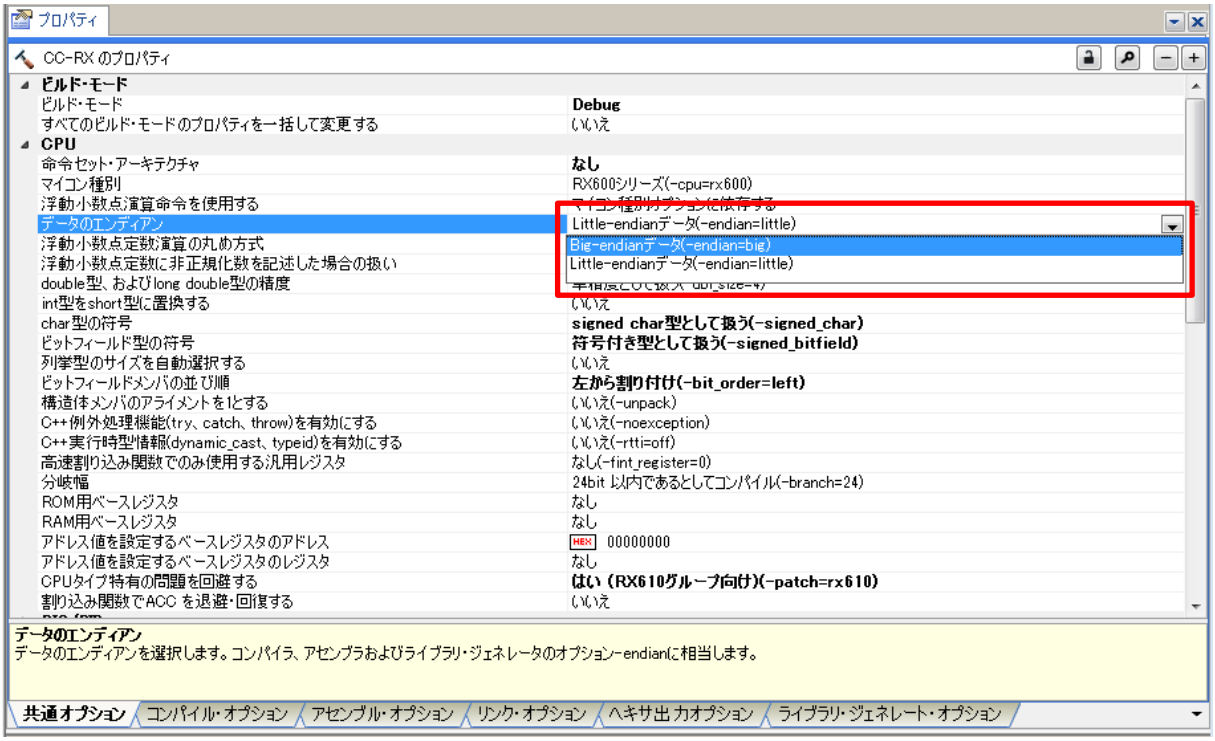

図 1-4

1.5　double 型のサイズ指定

H8 ファミリ用コンパイラでは double 型のサイズは 8byte です。対して RX ファミリ用コンパイラではデフォルトでは double 型のサイズは 4byte です。double 型のサイズが 8byte であることを前提に作成した H8 のプログラムを RX に移行するには、”dbl_size=8”オプションを指定します。

【書式】

dbl_size={4|8}　　:デフォルトは 4

[CS+でのオプション設定方法]

CC-RX（ビルド・ツール）プロパティの"共通オプション"タブ内で次のように設定します。

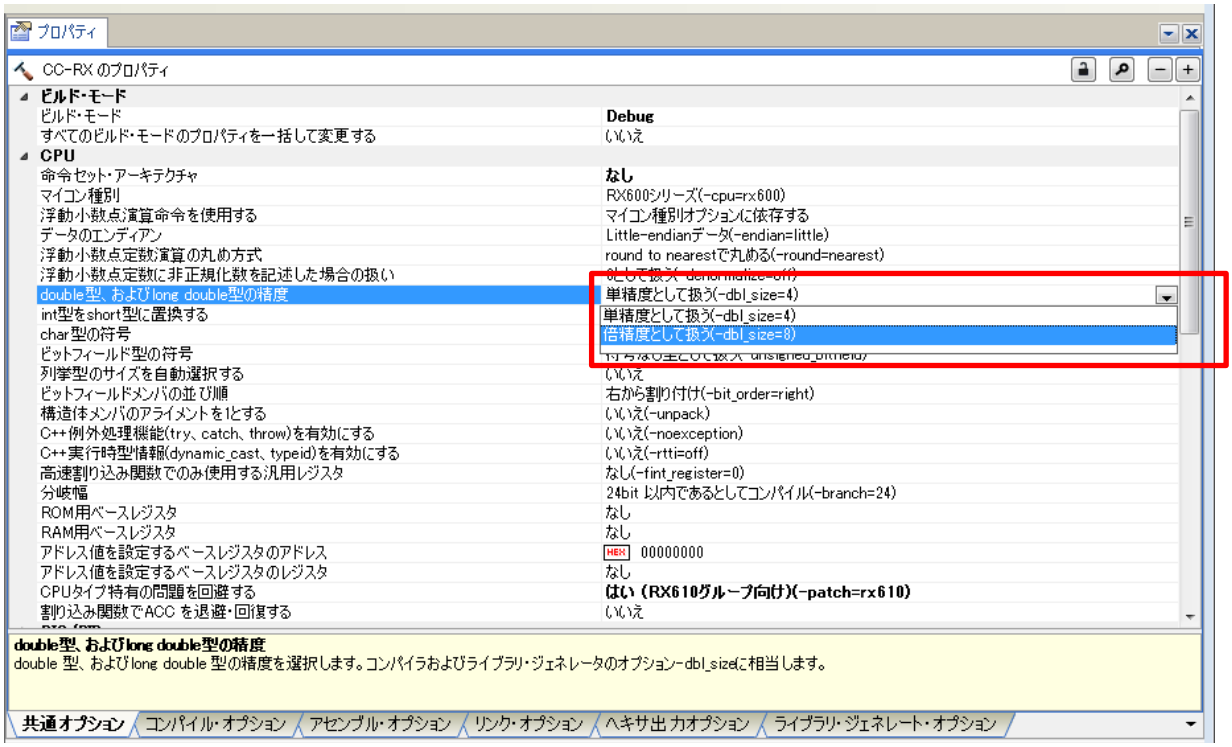

図 1-5

【注意事項】

H8 ファミリ用コンパイラの double=float オプションは、float 型および double 型のサイズを 4byte にします。long double 型は 8byte です。RX ファミリ用コンパイラの dbl_size=4 オプションは、float 型、double 型および long double 型のサイズを 4byte にします。

dbl_size=4 オプションが有効な時、浮動小数点関連の変換/ライブラリの結果が H8 ファミリ用コンパイラと結果が異なる場合があります。

1.6　int 型のサイズ違いへの対応

H8 ファミリ用コンパイラでは int 型のサイズは 2byte です。対して RX ファミリ用コンパイラでは int 型のサイズはデフォルトでは 4byte です。int 型のサイズが 2byte であることを前提に作成した H8 のプログラムを RX に移行するには、”int_to_short”オプションを指定します。

【書式】

int_to_short

[CS+でのオプション設定方法]

CC-RX（ビルド・ツール）プロパティの"共通オプション"タブ内で次のように設定します。

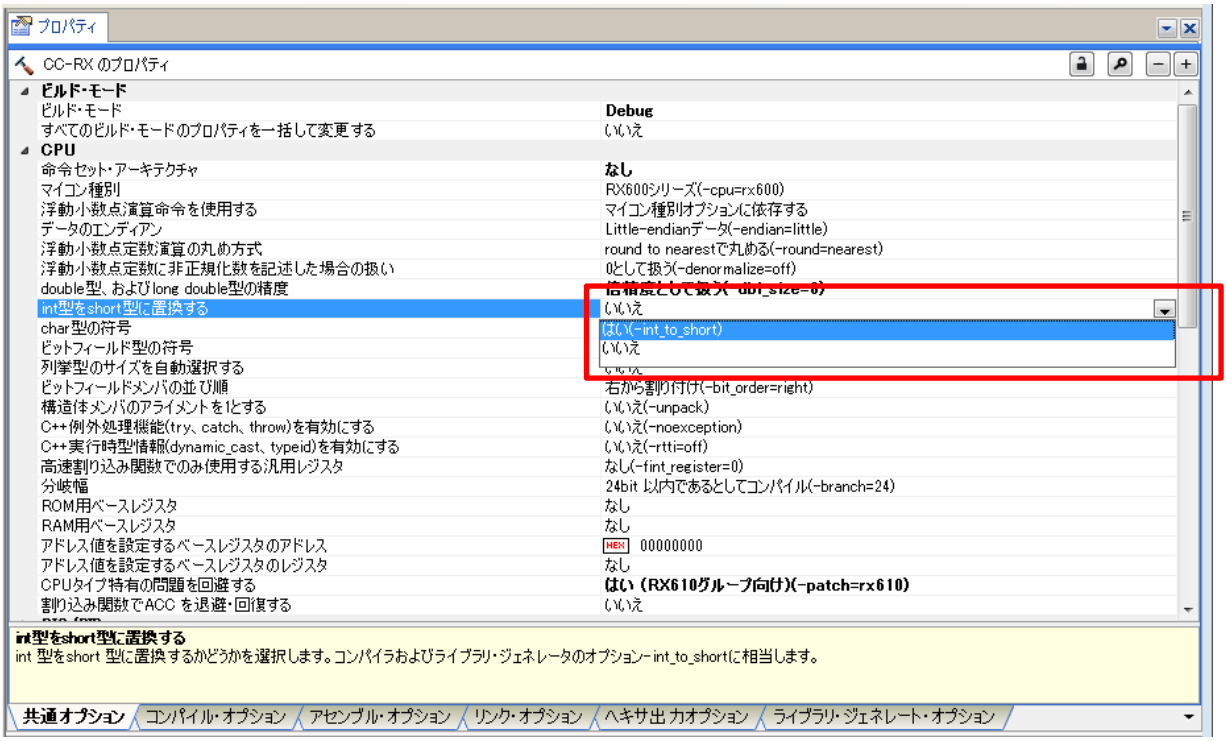

図 1-6

【注意事項】

ソースファイル内の int を short に置き換えてコンパイルしますが、limits.h の INT_MAX、INT_MIN、UINT_MAX のマクロの値は変わりません。

比較式などの汎整数拡張を行う場合は、4 バイト型に拡張して評価します。

本オプションは、lang=cpp オプションまたは、lang=ecpp オプションと組み合わせて指定はできません。組み合わせた場合、int_to_short オプションは無効になります。

2. 言語仕様

本章は、RX 移行時に変更が必要な言語仕様について説明します。

表 2-1　言語仕様一覧

No	機能	参照
1	char 型の符号	2.1
2	ビットフィールドメンバの符号	2.2
3	エンディアン	2.3
4	double 型のサイズ	2.4
5	int 型のサイズ	2.5
6	asm ブロック	2.6

2.1　char 型の符号

H8 ファミリ用コンパイラでは符号指定のない char 型は、符号ありの signed char 型として扱います。対して RX ファミリ用コンパイラではデフォルトでは符号なしの unsigned char 型として扱います。char 型が signed char 型であることを前提に作成した H8 のプログラムを、RX に移行すると正しく動作しない場合があります。

【例】char 型の符号の有無で動作が異なる記述例

ソースコード

```
char a = -1;

void main(void)
{
  if (a < 0) {
    // char 型が符号ありで’a’を負数と解釈し、条件式は成立(H8)
  } else {
    // char 型が符号なしで’a’を正数と解釈し、条件式は不成立(RX)
  }
}
```

char 型が符号ありの signed char 型であることを前提に作成したプログラムを RX に移行するには、”signed_char”オプションを指定します。オプション指定については「1.1 char 型の符号指定」を参照してください。

2.2　ビットフィールドメンバの符号

H8 ファミリ用コンパイラでは符号指定のないビットフィールドメンバは、符号ありの型として扱います。対して RX ファミリ用コンパイラではデフォルトでは符号なしの型として扱います。

符号指定のないビットフィールドメンバが符号ありの型であることを前提に作成した H8 のプログラムを、RX に移行すると正しく動作しない場合があります。

【例】ビットフィールドメンバの符号の有無で動作が異なる記述例

ソースコード

```
struct S {
  int a : 15;
} s = { -1 };

void main(void)
{
  if (s.a < 0) {
    // ビットフィールドメンバが符号ありで’s.a’を負数と解釈し、条件式は成立(H8)
  } else {
    // ビットフィールドメンバが符号なしで’s.a’を正数と解釈し、条件式は不成立(RX)
  }
}
```

符号指定のないビットフィールドメンバが符号ありの型であることを前提に作成したプログラムを RX に移行するには、”signed_bitfield”オプションを指定します。オプション指定については「1.2 ビットフィールドメンバの符号指定」を参照してください。

2.3　エンディアン

H8 ファミリ用コンパイラではデータのバイト並びが big endian になります。対して RX ファミリ用コンパイラではデフォルトでは little endian になります。

データのバイト並びが big endian であることを前提に作成した H8 のプログラムを、RX に移行すると正しく動作しない場合があります。

【例】エンディアンの差異で動作が異なる記述例

ソースコード

```
typedef union{
  short data1;
  struct {
    unsigned char upper;
    unsigned char lower;
  } data2;
} UN;

UN u = { 0x7f6f };

void main(void)
{
  if (u.data2.upper == 0x7f && u.data2.lower == 0x6f) {
    // データバイトの並びが big endian の場合(H8)
  } else {
    // データバイトの並びが little endian の場合(RX)
  }
}
```

データのバイト並びが big endian であることを前提に作成したプログラムを RX に移行するには、”endian=big”オプションを指定します。オプション指定については「1.4 エンディアン指定」を参照してください。

2.4　double 型のサイズ

H8 ファミリ用コンパイラでは double 型のサイズは 8byte です。対して RX ファミリ用コンパイラではデフォルトでは double 型のサイズは 4byte です。double 型のサイズが 8byte であることを前提に作成した H8 のプログラムを、RX に移行すると正しく動作しない場合があります。

【例】double 型のサイズの差異で動作が異なる記述例

ソースコード

```
double d1 = 1E30;
double d2 = 1E20;

void main(void)
{
  d1 = d1 * d1; // double 型のサイズが 4byte の場合、d1 * d1 がオーバフローする
  d2 = d2 * d2; // double 型のサイズが 4byte の場合、d2 * d2 がオーバフローする

  if (d1 > d2) {
    // double 型のサイズが 8byte の場合、正常な大小比較が行われる(H8)
  } else {
    // double 型のサイズが 4byte の場合、d1,d2 ともにオーバフローしているため大小比較が成立しない(RX)
  }
}
```

double 型のサイズが 8byte であることを前提に作成したプログラムを RX に移行するには、”dbl_size=8”オプションを指定します。オプション指定については「1.5 double 型のサイズ指定」を参照してください。

2.5　int 型のサイズ

H8 ファミリ用コンパイラでは int 型のサイズは 2byte です。対して RX ファミリ用コンパイラではデフォルトでは int 型のサイズは 4byte です。int 型のサイズが 2byte であることを前提に作成した H8 のプログラムを、RX に移行すると正しく動作しない場合があります。

【例】int 型のサイズの差異で動作が異なる記述例

ソースコード

```
typedef union{
  long data;
  struct {
    int dataH;
    int dataL;
  } s;
} UN;

void main(void)
{
  UN u;
  u.s.dataH = 0;
  u.s.dataL = 1;

  if (u.data == 0) {
    // int 型のサイズが 4byte の場合(RX)
  } else {
    // int 型のサイズが 2byte の場合(H8)
  }
}
```

int 型のサイズが 2byte であることを前提に作成したプログラムを RX に移行するには、”int_to_short”オプションを指定します。オプション指定については「1.6 int 型のサイズ違いへの対応」を参照してください。

2.6　asm ブロック

H8 ファミリ用コンパイラは asm ブロックで C ソースプログラム中に、アセンブリ言語プログラムを記述することができます。RX ファミリ用コンパイラではこれに相当する機能がないため、asm ブロックを利用したプログラムを RX に移行する場合、対応が必要となります。

RX ファミリ用コンパイラには C ソースプログラム中にアセンブリ言語を記述する、アセンブリ記述関数の機能があります。asm ブロックで記述した内容をアセンブラ記述関数内に記述することで対応可能なケースがあります。

アセンブリ記述関数の詳細は、コンパイラユーザーズマニュアルを参照下さい。

【例】H8 の asm ブロックを利用したプログラムと RX でアセンブリ記述関数を利用したプログラム

H8 asm ブロック利用 ソースコード

C ソースコード

```
void func(void)
{
  __asm {
    NOP
  }
}
```

アセンブラソース展開コード

```
_func:
  NOP
  rts
```

RX アセンブリ記述関数利用 ソースコード

C ソースコード

```
#pragma inline_asm asm_nop
static void asm_nop(void)
{
  NOP
}

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

アセンブラソース展開コード

```
_func:
  NOP
  RTS
```

【注意事項】

・H8 は asm ブロック内に変数を記述することができますが、RX ではできません。

3. 移行時の最適化オプションの設定に関して

H8 と RX のコンパイラでは、最適化オプションの設定方法が異なります。H8 から RX へ移行して性能評価を実施する場合は、以下の最適化オプションの設定を参考にしてください。

各コンパイラの最適化オプションの設定と ROM サイズの比較

（測定プログラムは、次項に添付のサンプルプログラム）

H8SX	最適化 OFF	Size 優先	Speed 優先
	opt=0	opt=1	opt=1 speed
main()	0xB8	0x96	0x96
sort()	0xA2	0x5E	0x6E

※H8 ファミリ用コンパイラ V.7.00 Release 00 にて測定

RX	最適化 OFF	Size 優先			Speed 優先		
	optimize=0	optimize=1	optimize=2	optimize=max	optimize=1 speed	optimize=2 speed	optimize=max speed
main()	0xAC	0x80	0x76	0x76	0x80	0x76	0x76
sort()	0x92	0x47	0x51	0x53	0x4A	0x5D	0x5F

※RX ファミリ用コンパイラ V2.05.00 にて測定

RX ファミリ用コンパイラの最適化レベルの詳細は、コンパイラユーザーズマニュアルを参照下さい。

参考　サンプルソース

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void main(void);
void sort(long *a);
void change(long *a);

void main(void)
{
    long a[10];
    long j;
    int i;

    printf("### Data Input ###¥n");

    for( i=0; i<10; i++ ){
        j = rand();
        if(j < 0){
            j = -j;
        }
        a[i] = j;
        printf("a[%d]=%ld¥n",i,a[i]);
    }
    sort(a);
    printf("*** Sorting results ***¥n");
    for( i=0; i<10; i++ ){
        printf("a[%d]=%ld¥n",i,a[i]);
    }
    change(a);
}
```

```
void sort(long *a)
{
    long t;
    int i, j, k, gap;

    gap = 5;
    while( gap > 0 ){
        for( k=0; k<gap; k++){
            for( i=k+gap; i<10; i=i+gap ){
                for(j=i-gap; j>=k; j=j-gap){
                    if(a[j]>a[j+gap]){
                        t = a[j];
                        a[j] = a[j+gap];
                        a[j+gap] = t;
                    }else{
                        break;
                    }
                }
            }
        }
        gap = gap/2;
    }
}
```

```
void change(long *a)
{
    long tmp[10];
    int i;
    for(i=0; i<10; i++){
        tmp[i] = a[i];
    }
    for(i=0; i<10; i++){
        a[i] = tmp[9 - i];
    }
}
```

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2009/10/1	-	初版発行
2.00	2016/11/30	-	移行先を CS+,CC-RXV2 に変更

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 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・システムの設計において、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て、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。
2. 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。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。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。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   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、産業用ロボット等
   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、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図し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損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の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。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、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、社会的損害等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、お客様の機器・システムとしての出荷保証を行ってください。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、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の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。
9. 本資料に記載されている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・システムに使用することはできません。また、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に使用しないでください。当社製品または技術を輸出する場合は、「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。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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禁じます。

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。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/