RX 開発環境移行ガイド

M16C から RX への移行（コンパイラ編）
（High-performance Embedded Workshop,NC30WA→CS+,CC-RX）

RJJ06J0078-0200
Rev.2.00
2016.11.30

要旨

本ドキュメントでは、ルネサス製 M16C ファミリ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、M16C 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。

M16C 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

M16C ファミリ用コンパイラのオプションの中に RX ファミリ用コンパイラのデフォルトオプションと仕様が異なるオプションが存在します。M16C 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	M16C オプション	RX オプション	参照
1	列挙型のサイズ変更指定	fchar_enumerator	auto_enum	1.1
2	double 型のサイズ指定	fdouble_32	dbl_size	1.2
3	int 型のサイズ指定	－	int_to_short	1.3

1.1　　列挙型のサイズ変更指定

M16C ファミリ用コンパイラで”fchar_enumerator”オプションを指定すると列挙子の型を int 型ではなく unsigned char 型で扱います。

RX ファミリ用コンパイラで同様の効果を得るには、”auto_enum”オプションを指定します。但し、列挙子の最小値が 0 以上かつ、最大値が 255 以下の場合のみ列挙子の型は unsigned char 型になり、それ以外の場合は異なる型になります。列挙型のとりうる値と型の関係の詳細は、コンパイラユーザーズマニュアルを参照下さい。

【書式】

auto_enum

[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。

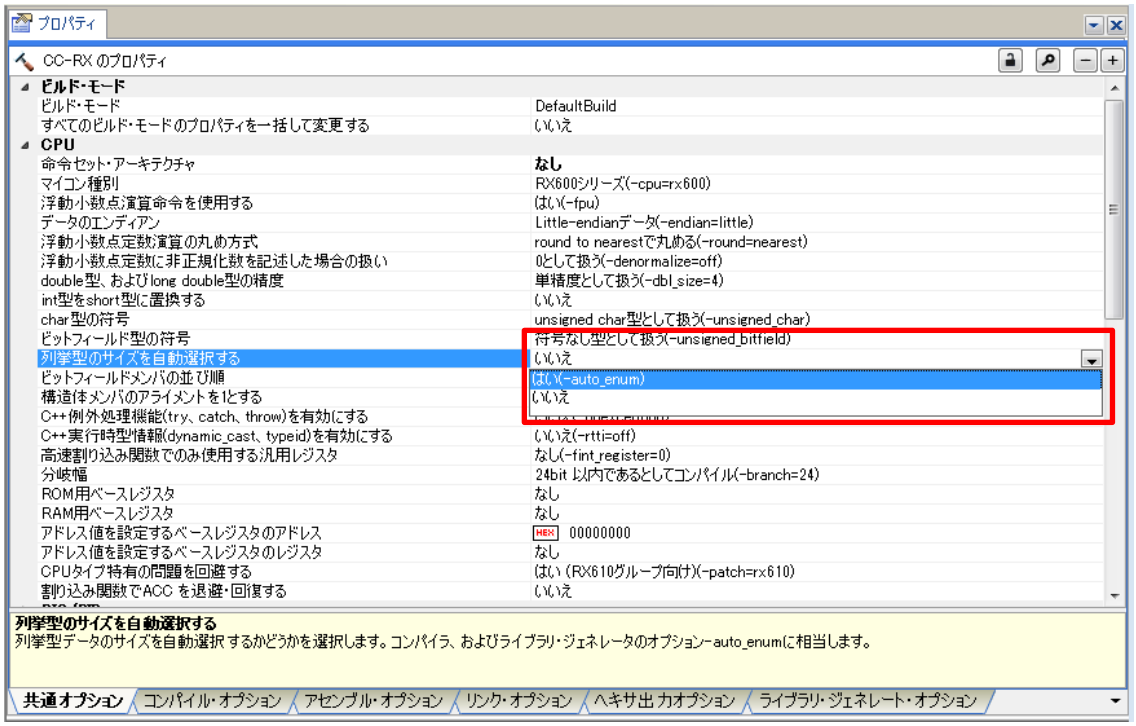


図 1-1

1.2　double 型のサイズ指定

M16C ファミリ用コンパイラで”fdouble_32”オプション指定がない場合、double 型のサイズを 8byte(倍精度)として扱います。

RX ファミリ用コンパイラで同様の解釈をするには、”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。

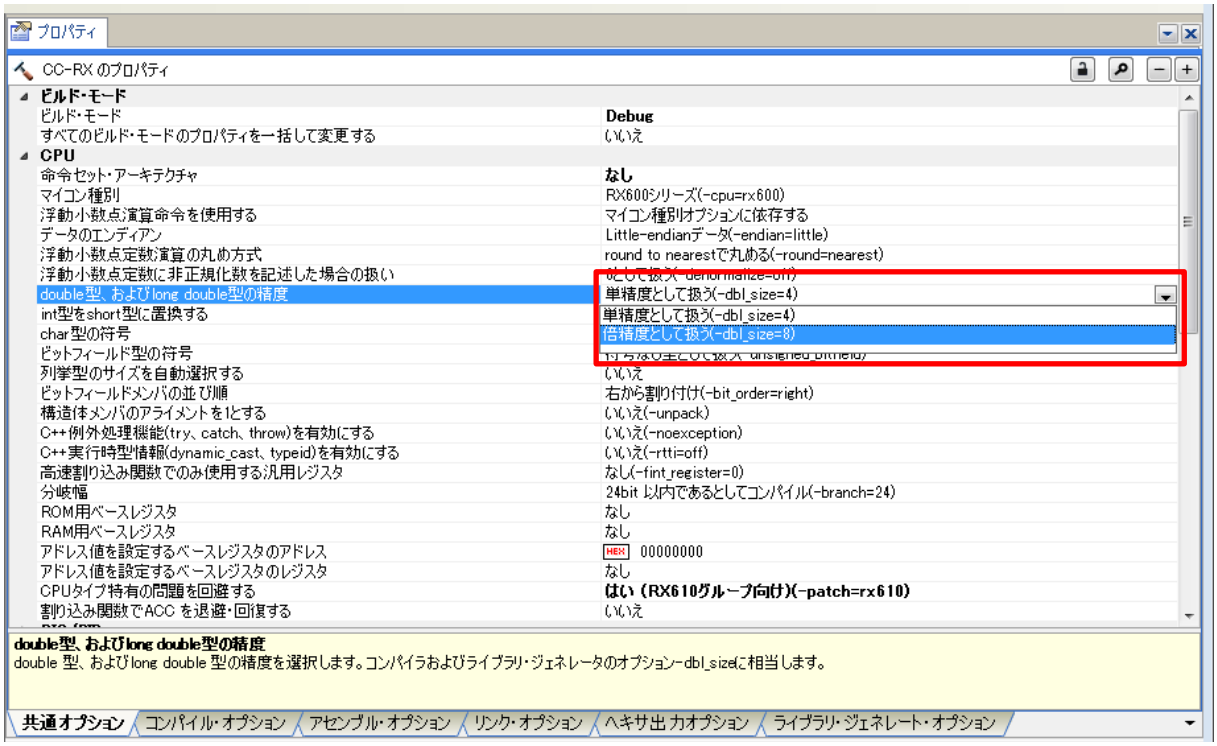


図 1-2

1.3　int 型のサイズ指定

M16C ファミリ用コンパイラでは、int 型のサイズを 2byte として扱います。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 4byte です。

int 型のサイズが 2byte（汎整数型変換は除きます）であることを前提に作成した M16C 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。

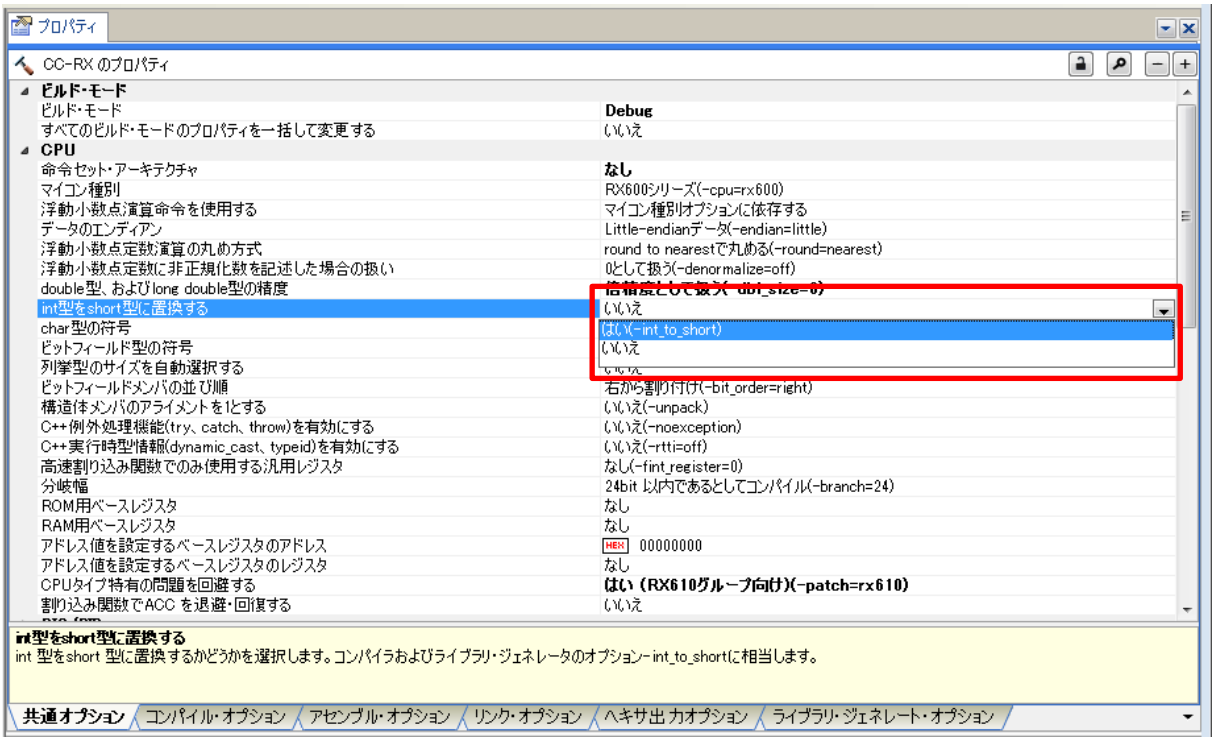


図 1-3

2. 言語仕様

本章は RX 移行時に変更が必要である可能性の高い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	int 型のサイズ	2.1
2	double 型のサイズ	2.2
3	char 型の汎整数拡張	2.3
4	構造体のメンバ配置	2.4
5	inline キーワード	2.5
6	#pragma STRUCT	2.6
7	#pragma BITADDRESS	2.7
8	#pragma ROM	2.8
9	#pragma PARAMETER	2.9
10	asm 関数	2.10

2.1　int 型のサイズ

M16C 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 4byte です。int 型のサイズが 2byte であることを前提に作成した M16C 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(M16C)
  }
}
```

int 型のサイズが 2byte（汎整数型変換は除きます）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「1.3 int 型のサイズ指定」を参照してください。

2.2 double 型のサイズ

M16C 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 double 型のサイズはデフォルトで 4byte です。double 型のサイズが 8byte であることを前提に作成した M16C 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;
  d2 = d2 * d2;
  if (d1 > d2) {
    // double 型のサイズが 8byte の場合(M16C)
  } else {
    // double 型のサイズが 4byte の場合(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「1.2 double 型のサイズ指定」を参照してください。

2.3 char 型の汎整数拡張

M16C ファミリ用コンパイラでは char 型データ(unsigned char, signed char を含む)を評価する時に、int 型に拡張していません。対して RX ファミリ用コンパイラでは char 型データを評価する時に、必ず int 型に拡張します。M16C で本件を前提に作成したプログラムを RX に移行すると、M16C 同様の動作をしない場合があります。

【例】char 型演算の汎整数拡張仕様の差異で動作が異なる記述例

ソースコード

```
char c1;
char c2 = 200;
char c3 = 200;
void main(void)
{
  // M16C では c2 + c3 の演算で表現できる最大値をオーバフローするため予期しない結果となる
  c1 = (c2 + c3) / 2;

  if (c1 == 200) {
    // char 型を int 型に拡張して評価する場合(RX)
  } else {
    // char 型を int 型に拡張せず評価する場合(M16C)
  }
}
```

【補足】

M16C ファミリ用コンパイラは char 型データ(unsigned char, signed char を含む)を評価する時に、int 型に拡張するオプションを用意しています。以下 2 点のオプションがそれにあたります。以下のオプションのいずれかが指定されている場合、本項で説明した汎整数拡張仕様による問題は発生しません。

・-fansi

・-fextend_to_int

2.4　構造体のメンバ配置

M16C ファミリ用コンパイラは構造体のメンバ配置をメンバデータの出現順にアライメント数 1 で配置します。対して RX ファミリ用コンパイラは構造体のメンバ配置をメンバデータの出現順に、メンバの最大のアライメント数に従って配置します。M16C の構造体配置であることを前提に作成したプログラムを RX に移行した場合、正しく動作しない場合があります。このような場合は、“pack”オプションを指定することで、構造体メンバのアライメント数を 1 とすることができます。アライメント数が 1 となった構造体は空き領域が作られなくなります。

また、#pragma pack の指定でも構造体のアライメント数を指定することができます。オプションと#pragma が同時に指定された場合は、#pragma の指定を優先します。

機能の詳細は、コンパイラユーザーズマニュアルを参照ください。

【書式】

pack　　：デフォルトは unpack

unpack

[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。

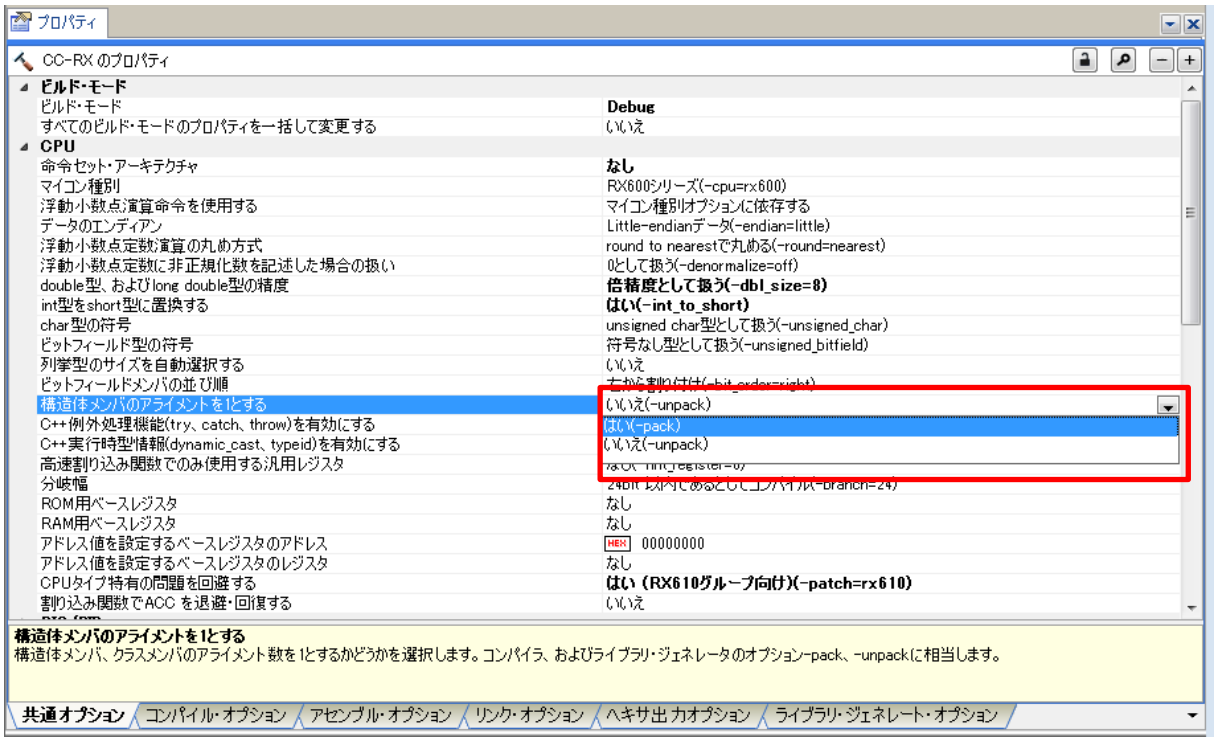


図 2-1

2.5 inline キーワード

M16C ファミリ用コンパイラは inline キーワードをサポートしています。対して RX ファミリ用コンパイラの ANSI 規格 C89 モードでは inline キーワードをサポートしていないため、inline キーワードを使用したプログラムはコンパイルエラーとなります。以下の 2 つのうちいずれかの方法で対応可能です。

- inline キーワードの記述を#pragma inline へ変更する。
- ANSI 規格 C99 でビルドする。

【例】

M16C の inline キーワードを RX の#pragma inline に移行したプログラム

ソースコード(inline キーワード使用)

```
inline static int func(int a, int b)
{
  return (a + b) / 2;
}
int x;
void main(void)
{
  x = func(10, 20);
}
```

ソースコード(#pragma inline 使用)

```
#pragma inline(func)
static int func(int a, int b)
{
  return (a + b) / 2;
}
int x;
void main(void)
{
  x = func(10, 20);
}
```

ANSI 規格 C99 でビルドする場合は“lang=c99”オプションを指定します。

【書式】

lang={c|cpp|ecpp|c99}

[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"コンパイル・オプション"タブ内で次のように設定します。

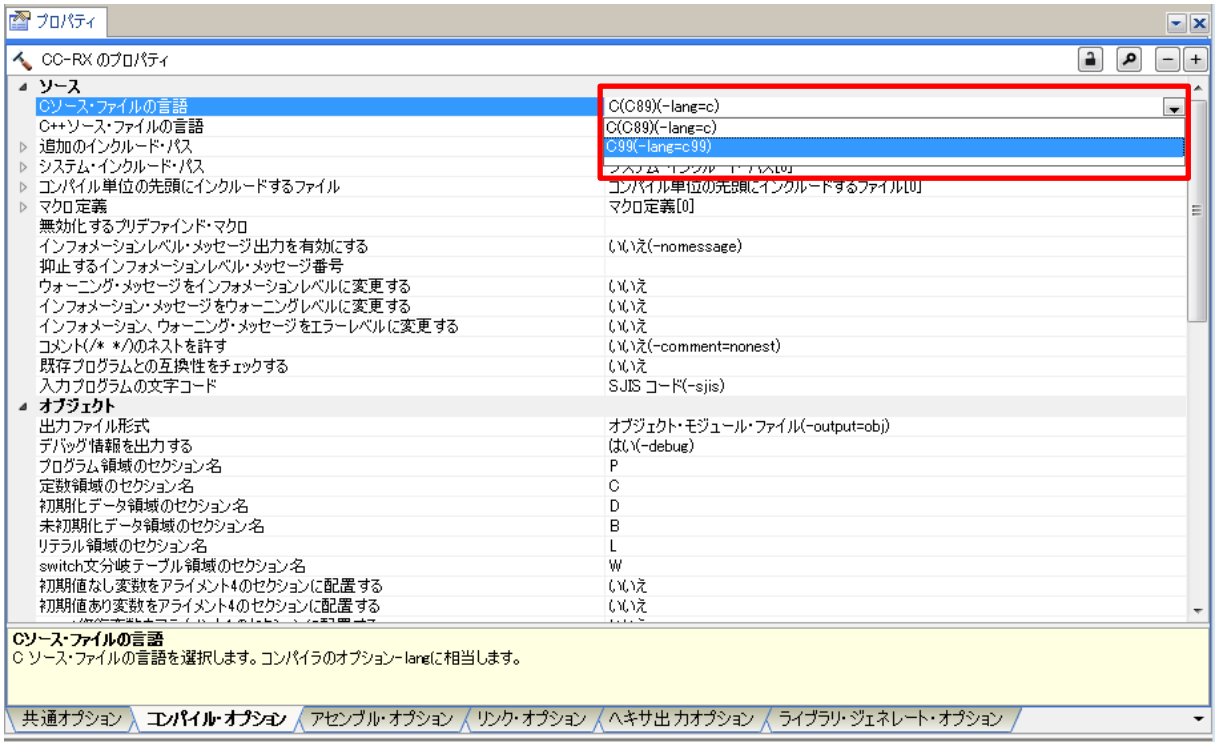


図 2-2

【注意事項】

1. ANSI 規格 C99、C++の inline キーワード、#pragma inline の仕様は以下のように異なります。

表 2-2　inline 仕様

	デフォルトの リンケージ	inline 展開できなかった 場合のリンケージ	宣言による リンケージへの影響
C99 inline キーワード	内部リンケージ	内部リンケージ	extern を指定した場合 外部リンケージ
C++ inline キーワード	内部リンケージ	内部リンケージ	extern を指定した場合 コンパイルエラー
#pragma inline	外部リンケージ	外部リンケージ	static を指定した場合 内部リンケージ

2. inline キーワードと#pragma inline の展開は異なります。#pragma inline は強制展開です。

2.6　#pragma STRUCT

M16C ファミリ用コンパイラは、#pragma STRUCT で構造体のパック禁止や、構造体メンバの並び替えを行うことができます。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、#pragma STRUCT 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。

(1) 構造体のパック禁止

M16C ファミリ用コンパイラはデフォルトで構造体メンバをアライメント数 1 で配置します。そのため構造体のサイズが奇数バイトになることがあります。構造体のサイズを偶数バイトにしたい場合、#pragma STRUCT unpack を指定します。指定により構造体のサイズが奇数バイトの場合は 1 バイトのパディングを入れ、構造体のサイズを偶数バイトにします。

RX ファミリ用コンパイラで同様の効果を得るには以下の 2 点を実施する必要があります。

- #pragma pack 指定により構造体のアライメント数を 1 にする。
- アライメント数を 1 にした結果、構造体のサイズが奇数になった場合、調整用の 1byte のダミーメンバをユーザが挿入する。

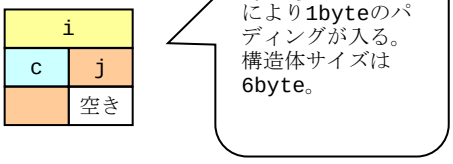
RX ファミリ用コンパイラで構造体のアライメント数を 1 にするには、#pragma pack 指定の他に”pack”オプションを指定する方法もあります。機能の詳細は”2.4 構造体のメンバ配置”を参照してください。

【例】

M16C の#pragma STRUCT unpack プログラムと RX に移行したプログラム

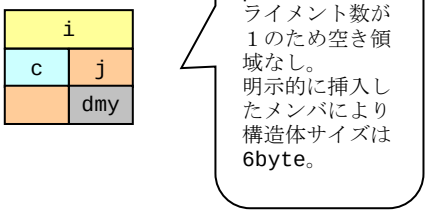
#pragma STRUCT unpack 指定ソースコード

```
#pragma STRUCT s unpack
struct s {
    short  i;
    char  c;
    short  j;
} ss;
```



RX ソースコード

```
#pragma pack
struct s {
    short  i;
    char c;
    short  j;
    char dmy;    // 偶数バイトのサイズとするため
                 // 1byte のダミーメンバを挿入する
} ss;
```



(2) 構造体メンバの並び替え

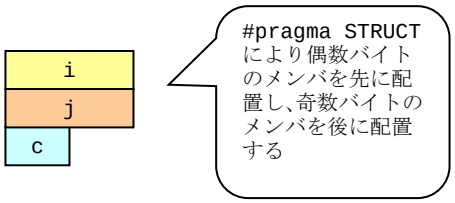
RX ファミリ用コンパイラでは構造体メンバを並び替える機能がありません。メンバの並び替えを行うにはプログラムを変更する必要があります。

【例】

M16C の#pragma STRUCT arrange プログラムと RX に移行したプログラム

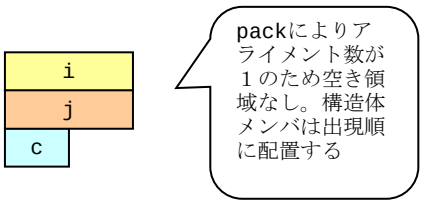
#pragma STRUCT arrange 指定ソースコード

```
#pragma STRUCT s arrange
struct s {
    short  i;
    char  c;
    short  j;
} ss;
```



RX ソースコード

```
#pragma pack
struct s {
    short  i;
    short  j;       // メンバ'j'と'c'の配置を変更する
    char c;
} ss;
```



【補足】

M16C の#pragma STRUCT unpack と RX の#pragma unpack は意味が異なるので注意してください。

・#pragma STRUCT unpack (M16C)

指定された構造体のサイズを偶数バイトに調整します。

・#pragma unpack (RX)

指定された構造体のアライメント数をメンバの最大のアライメント数と同じにします。

2.7 #pragma BITADDRESS

M16C ファミリ用コンパイラは#pragma BITADDRESS で指定した絶対アドレスの指定したビット位置に、変数を割り付けることができます。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、#pragmaBITADDRESS を使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。

【例】

M16C の#pragma BITADDRESS で 100 番地ビット番号 1 に 1 をセットするプログラムと RX に移行したプログラム

#pragma BITADDRESS 指定ソースコード

```
#pragma BITADDRESS io 1, 100H
_Bool io;

void main(void)
{
  io = 1;
}
```

RX ソースコード

```
struct bit_address {
  unsigned char b0:1;
  unsigned char b1:1;
  unsigned char b2:1;
  unsigned char b3:1;
  unsigned char b4:1;
  unsigned char b5:1;
  unsigned char b6:1;
  unsigned char b7:1;
};

#define io (((struct bit_address*)0x100)->b1)

void main(void)
{
  io = 1;
}
```

【補足】

M16C ファミリ用コンパイラで_Bool キーワードを使用したプログラムを RX ファミリ用コンパイラの ANSI 規格 C89 でビルドした場合、_Bool キーワードは規格外でビルド時にエラーとなります。RX ファミリ用コンパイラは ANSI 規格 C99 で_Bool キーワードをサポートしています。

2.8　#pragma ROM

M16C ファミリ用コンパイラは#pragma ROM で指定した変数を rom セクションに配置します。RX ファミリ用コンパイラはこれに相当する機能がないため、#pragma ROM を利用したプログラムを RX に移行する場合、const 修飾子を用いて rom セクションに配置する必要があります。

【例】

M16C の#pragma ROM で変数’i’を rom セクションに配置するプログラムと RX に移行したプログラム

#pragma ROM 指定ソースコード

```
#pragma ROM i
unsigned short i;
```

アセンブラ展開コード

```
  .SECTION rom_FE,ROMDATA,align
  .glb _i
_i:
  .byte 00H
  .byte 00H
```

RX ソースコード

```
const unsigned short i;    // const キーワードを付加
```

アセンブラ展開コード

```
  .glb _i
  .SECTION C_2,ROMDATA,ALIGN=2
_i:
  .word 0000H
```

2.9　#pragma PARAMETER

M16C ファミリ用コンパイラは#pragma PARAMETER で、引数をレジスタに格納して渡すアセンブラ関数を宣言できます。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、#pragma PARAMETER 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は、任意のレジスタに引数を格納する指定ができません。関数呼び出しの引数インタフェースは、コンパイラの生成規則に従う必要があります。RX 移行時に#pragma PARAMETER で指定されたアセンブラ関数の引数インタフェースを、コンパイラの生成規則に従い変更してください。

引数インタフェースの詳細は、コンパイラユーザーズマニュアルを参照下さい。

【例】

M16C の#pragma PARAMETER を利用したプログラムと RX でアセンブラ関数と引数インタフェースを合わせたプログラム

#pragma PARAMETER 指定ソースコード

C ソースコード

```
int asm_func(unsigned int, unsigned int);
#pragma PARAMETER asm_func(R0, R1)

void main(void)
{
  int i, j;

  i = 0x7FFD;
  j = 0x007F;

  // アセンブラ関数の呼び出し
  // ’i’は R1、’j’は R0 に格納
  asm_func( i, j );
}
```

アセンブラソースコード

```
_asm_func:
   ‘i’は R1、’j’は R0 に格納して渡ることが
  前提のコード
    …
  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

```
int asm_func(unsigned int, unsigned int);
void main(void)
{
  int i, j;

  i = 0x7FFD;
  j = 0x007F;

  // アセンブラ関数の呼び出し
  // コンパイラの引数インタフェースに従い
  // ‘i’は R1、’j’は R2 に格納
  asm_func( i, j );
}
```

アセンブラソースコード

```
_asm_func:
   ‘i’は R1、’j’は R2 に格納して渡されることを
  前提にコードを変更する
    …
  RTS
```

2.10　asm 関数

M16C ファミリ用コンパイラは asm 関数を使って、C ソースプログラム中にアセンブリ言語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 関数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 関数で記述した内容を、アセンブラ記述関数内に記述することで対応が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】

M16C の asm 関数を利用したプログラムと RX でアセンブラ記述関数を利用したプログラム（コード内容は等価ではありません。移行例として活用ください）

M16C asm 関数 指定ソースコード

C ソースコード

```
void func(void)
{
  asm("FSET I");
}
```

アセンブラソース展開コード

```
_func:
  FSET I
  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 interrupt_flag
static void interrupt_flag(void)
{
  MOV.L #00010000H,R5
  MVTC R5,PSW
}

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

アセンブラソース展開コード

```
_func:
  MOV.L #00010000H,R5
  MVTC R5,PSW
  RTS
```

【注意事項】

- M16C は asm 関数に変数を記述することができますが、RX ではできません。
- M16C は最適化を部分的に抑止する手段の一つとしてダミーの asm 関数を使えますが、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

M16C 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。M16C 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）

M16C	最適化 OFF	Size 優先		Speed 優先	
	O0	O3 OR	OR_MAX	O3 OS	OS_MAX
main()	0xE6	0x99	0x98	0xB4	0x358
sort()	0xF6	0x92	0x91	0x94	0x94

※M16C ファミリ用コンパイラ V.6.00 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/

すべての商標および登録商標は，それぞれの所有者に帰属します。

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

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/

© 2016 Renesas Electronics Corporation. All rights reserved.
Colophon 4.0