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。

RX ファミリ 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＞拡張機能編

本ドキュメントでは、RX ファミリ C/C++コンパイラ V.1 において、サイズもしくは実行速度の性能向上が期待できる拡張機能(#pragma) と便利な#pragma について説明します。

目次

1　実行速度の性能向上が期待できる拡張機能

本章では、サイズもしくは実行速度の性能向上が期待できる拡張機能(#pragma)について説明します。
サイズもしくは実行速度の性能向上が期待できる拡張機能を表 1-1に示します。

表 1-1 性能向上が期待できる拡張機能

No	#pragma	機能	サイズ	実行速度	参照
1	#pragma inline	**関数のインライン展開指定**	×	◎	1.1
2	#pragma inline_asm	**アセンブリ記述関数のインライン展開**	×	○	1.2

◎　**特に有効である**
○　**有効である**
×　**劣化する**

なお、本ドキュメントのアセンブリ言語展開コードは、“output=src” および “cpu=rx600” を指定して取得しています。
ただし、“cpu” オプションにより、アセンブリ言語展開コードが異なる場合があります。また、アセンブリ言語展開コードは今後のコンパイラ改善などにより変わる可能性があります。

1.1　関数のインライン展開指定

インライン展開とは、関数呼び出しの位置に呼び出し先のプログラムを展開する最適化処理で、関数呼び出しに伴うオーバーヘッドの削減により、速度向上が期待できます。特にループ内で呼ばれる関数を展開すると、呼び出し回数が多いために大きな効果が期待できます。なお、コンパイラはインライン展開された後のコードに対して最適化を実施します。そのため、大きな関数をインライン展開するとプログラムサイズが大きくなり、コンパイラの最適化効率が低下する恐れがあります。インライン展開は呼び出し頻度が高い小さな関数に指定すると効果的です。

【書式】

#pragma inline [(]<関数名>[,…][)]

#pragma inline は、関数本体の定義の前に指定してください。
#pragma inline で指定した関数に対しても外部定義を生成します。外部定義が不要な場合は、関数の宣言に static を指定してください。static を指定した関数がインライン展開された場合は、当該関数の実体が生成されないため、サイズの削減が期待できます。
inline オプションを指定すると、指定されたサイズ分自動インライン展開を行います。自動インライン展開のサイズの限度を超える場合でも、#pragma inline 指定した関数はインライン展開を行います。
なお、#pragma inline の指定が行われていても、以下の条件を満たす場合はインライン展開されませんので、注意してください。
― #pragma inline 指定より前に関数の定義がある。
― 可変パラメータを持つ関数である。
― 関数のアドレスを介して呼び出しを行っている。

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

    func_p = func;

    x=func_p(10,20);
}
```

関数アドレスを関数型ポインタに代入し、その関数型ポインタを使用して関数コールをすると、インライン展開指定関数でも、インライン展開されない。

【例】インライン展開のイメージ

ソースコード

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

展開イメージ

```
int x;
void main(void)
{
    int func_result;
    {
        int a_1=10, b_1=20;
        func_result=(a_1+b_1)/2;
    }
    x=func_result;
}
```

1.2 アセンブリ記述関数のインライン展開

C 言語でサポートしていない CPU 命令を使用したい場合や、C 言語で記述するよりもアセンブラで記述し、性能を向上させたい場合、アセンブラ埋め込みインライン展開機能を用いると便利です。C ソースファイルの関数に対して"#pragma inline_asm"でアセンブラ記述関数であることを宣言すると、関数内にアセンブリ記述を行うことができます(これをインラインアセンブリ関数と呼びます)。

【書式】

#pragma inline_asm [(]<関数名> [,…][)]

インラインアセンブリ関数を記述する際には、以下の点に注意してください。

- ラベルはテンポラリラベルを使用する。
  (テンポラリラベル:‘？：’)
- 定義の最後に RTS(リターン)命令は記述しない。
- 保証レジスタの退避／回復をする。

もし、#pragma inline_asm で指定した関数でコンパイルエラーになった場合、コンパイラのデバッグ情報を出力していると、C ソースの行情報が出力されるため、ルネサス統合開発環境には C ソースのライン情報が表示されます。したがって、表示されるライン情報からは、エラーの原因となっているアセンブリプログラムの行にはジャンプできません。コンパイラのデバッグ情報の出力を止めると、ルネサス統合開発環境に表示されるライン情報は、アセンブリプログラムの行が出力されます。#pragma inline_asm で指定した関数をデバッグする場合は、コンパイラのデバッグ情報の指定を抑止することをお勧めします。

なお、関数間のインタフェースはC/C++コンパイラの生成規則に従ってください(表 1-2、表 1-3)。

表 1-2 C 言語プログラムでの引数割り付けの一般規則

割付規則		
レジスタで渡される引数		スタックで渡される引数
引数格納用レジスタ	対象の型	
R1～R4 のうち１つ	signed char、(unsigned)char、bool、(signed)short、unsigned short、(signed)int、unsigned int、(signed)long、unsigned long、float、double*1、long double*1、ポインタ、データメンバへのポインタ、リファレンス	• 引数の型がレジスタ渡しの対象の型以外のもの • 関数原型により可変個の引数を持つ関数として宣言しているもの*3 • R1～R4 のうち、まだ他の引数に割り当てられていないものの本数が、割り当てに必要な本数より少ない場合
R1～R4 のうち２つ	(signed)long long、unsigned long long、double*2、long double*2	
R1～R4 のうち、サイズを 4 で割った数	サイズが 4 の倍数の構造体、共用体、クラス型	

[注釈] *1 dbl_size=8 を指定しなかった場合です。
*2 dbl_size=8 を指定した場合です。
*3 関数原型により可変個の引数をもつ関数として宣言している場合、宣言の中で対応する型のない引数およびその直前の引数はスタック渡しになります。型のない引数は、2 バイト以下の整数は long 型に、float 型は double 型にそれぞれ変換して、全て境界調整数が 4 の引数として取り扱います。

表 1-3 C 言語プログラムでのリターン値の型と設定場所

リターン値の型	設定場所
singed char、(unsigned) char、(singed) short、unsigned short、(singed) int、unsigned int、(signed) long、unsigned long、float、double*2、long double*2、ポインタ、bool、リファレンス、データメンバへのポインタ	R1 signed char、(signed)short は符号拡張、(unsigned)char、unsigned short はゼロ拡張を行った結果を設定
double*3、long double*3、(signed)long long、unsigned long long	R1、R2 下位 4 バイトを R1 に、上位 4 バイトを R2 に設定
16 バイト以内かつ 4 の倍数であるサイズの構造体、共用体、クラス型	メモリイメージの先頭から 4 バイトずつ R1、R2、R3、R4 の順に設定
上記以外の構造体、共用体、クラス型	リターン値設定領域(メモリ)*1

[注釈] *1 関数のリターン値をメモリに設定する場合、リターン値はリターン値アドレスの指す領域に設定します。呼び出す側では、引数領域のほかにリターン値設定領域を確保し、そのアドレスを R15 に設定してから関数を呼び出します。
*2 dbl_size=8 を設定しなかった場合です。
*3 dbl_size=8 を指定した場合です。

【例 1】インラインアセンブリ関数内のアセンブリ記述例

ソースコード

```
/* Inline function definition */
/* FILE: inlasm.h */
#pragma inline_asm(rev4b)
static unsigned long rev4b(unsigned long p)
/* 関数はstatic で宣言*/
{
  ; 定義中のコメントは アセンブラの;（セミコロン）を使用する
    REVL R1,R1
  ; 最後にRTS 命令は記述しない
}
#pragma inline_asm(ovf)
static unsigned long ovf(void)
{
?:; インラインアセンブリ関数内ではテンポラリラベルを使用する
  ; 関数の呼び出し前後で保障が必要なレジスタは退避回復を行う
    PUSH.L R6
    MOV.L R1,R6
  ;   :
    CMP #1,R6
    BEQ.W ?-
    POP R6
}
```

2　その他の便利な拡張機能

本章では、性能向上以外に便利な拡張機能について説明します。便利な拡張機能を表 2-1に示します。

表 2-1 便利な拡張機能

No	#pragma	機能	参照
1	#pragma section	**セクションの切り替え指定**	2.1
2	#pragma bit_order	**ビットフィールドの並び順指定**	2.2
3	#pragma pack #pragma unpack #pragma packoption	**エラー! 参照元が見つかりません。**	**エラー! 参照元が見つかりません。**

2.1　セクションの切り替え指定

コンパイラの出力するセクション名を切り替えることができます。

例えば、あるモジュールを外付け RAM に、別のモジュールを内蔵 RAM に割り付けたい場合など、別々のアドレスに割り付けたい場合があります。分割したいセクションそれぞれに名称をつけます。そして、それぞれのセクションに対して配置したいアドレスをリンケージエディタで指定します。

#pragma section で、セクション名を指定しない場合、それ以降はデフォルトのセクション名が適用されます。

【書式】

#pragma section [<セクション種別>][△<変更セクション名>]
<セクション種別>:{ P | C | D | B |W}

切り替え後のセクション名を表 2-2、表 2-3に示します。

表 2-2 セクション切り替え機能とセクション名(セクション種別無)

	対象領域	指定方法	切り替え後
1	**プログラム領域**	#pragma section <XX>	P<XX>
2	**定数領域**		C<XX>
3	**初期化データ領域**		D<XX>
4	**未初期化データ領域**		B<XX>
4	**switch 文分岐テーブル領域**		W<XX>

表 2-3 セクション切り替え機能とセクション名(セクション種別有)

	対象領域	指定方法	セクション種別<X>	切り替え後
1	**プログラム領域**	#pragma section <X> <XX>	P	<XX>
2	**定数領域**		C	<XX>
3	**初期化データ領域**		D	<XX>
4	**未初期化データ領域**		B	<XX>
4	**switch 文分岐テーブル領域**		W	<XX>

“section” オプションで、デフォルトのセクション名を変更できます。

[“secsion”オプションの書式]

section = <sub>[,…]
<sub>:　{　P　=　<セクション名>　|
C　=　<セクション名>　|
D　=　<セクション名>　|
B　=　<セクション名>　|
W　=　<セクション名>　}

[ルネサス統合開発環境でのオプション設定方法]

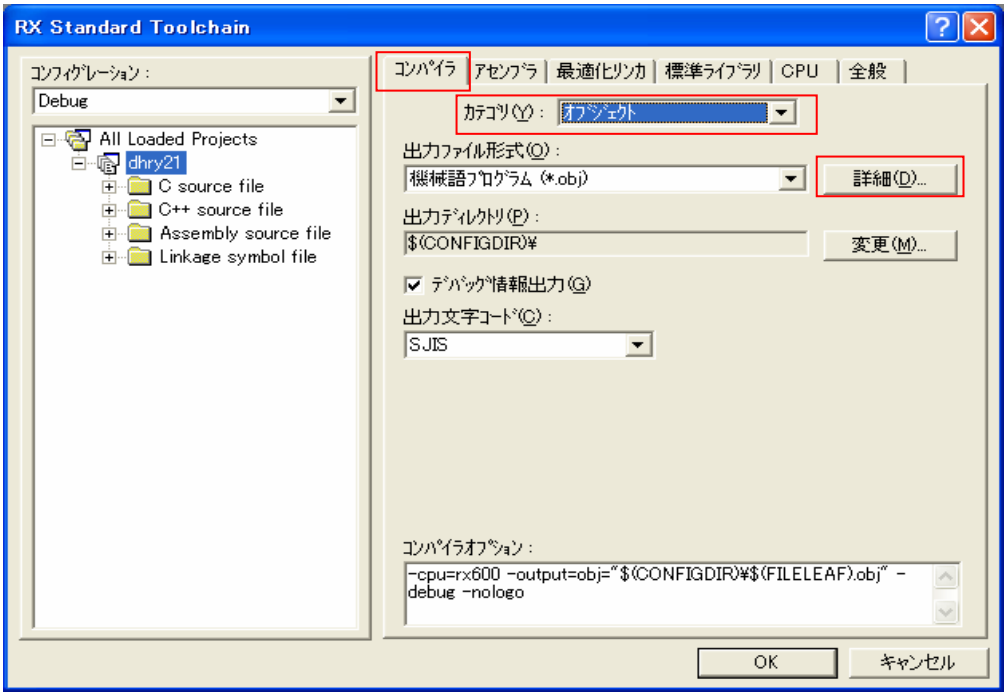


図 2-1

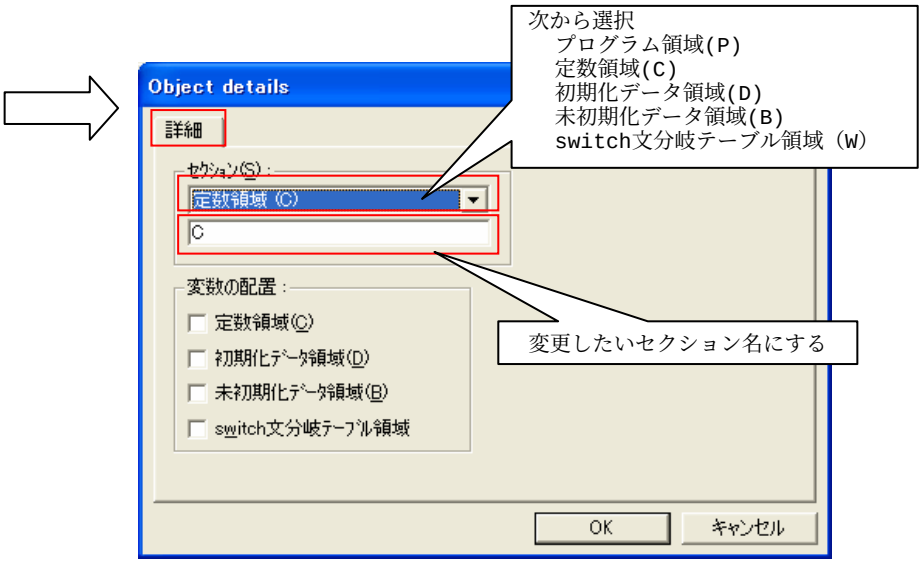


図 2-2

【例 1】

ソースコード

```
#pragma section abc

int a;          /* a はセクションBabc に割り付きます */
const int c=1; /* c はセクションCabc に割り付きます */
void f(void)    /* f はセクションPabc に割り付きます */
{
    a=c;
}

#pragma section

int b;         /* b はセクションB に割り付きます */
void g(void)  /* g はセクションP に割り付きます */
{
    b=c;
}
```

【例 2】

ソースコード

```
#pragma section P ABC
#pragma section B DEF

int a;          /* a はセクションDEF に割り付きます */
const int c=1; /* c はセクションC に割り付きます */
void f(void)    /* f はセクションABC に割り付きます */
{
    a=c;
}

#pragma section

int b;         /* b はセクションB に割り付きます */
void g(void)  /* g はセクションP に割り付きます */
{
    b=c;
}
```

【例 3】

section=P=PX,C=CX,B=BX と指定した場合

ソースコード

```
#pragma section abc

int a;           /* a はセクションBXabc に割り付きます */
const int c=1;  /* c はセクションCXabc に割り付きます */
void f(void)     /* f はセクションPXabc に割り付きます */
{
    a=c;
}

#pragma section

int b;          /* b はセクションBX に割り付きます */
void g(void)  /* g はセクションPX に割り付きます */
{
    b=c;
}
```

2.2 ビットフィールドの並び順指定

#pragma bit_order 指定を使用すると、ビットフィールドの並び順を変更することができます。マイコンによってはビットフィールドの並び規則が違うものがあります。本機能を使用すると他のマイコンで動作していたプログラムの移植性が向上します。

ファイルごとのビットフィールドの並び順指定はオプションでも指定できます。オプション、#pragma 同時に指定された場合は、#pragma の指定を優先します。

【書式】

#pragma bit_order [{left|right}]

left を指定した場合は上位ビット側から、right を指定した場合は下位ビット側から、それぞれメンバが割り付けられます。デフォルトの設定は下位ビット側から割り付けとなります。#pragma bit_order の指定で、left|right を省略すると、その行以降はオプションに従います。

【例 1】

#pragma bit_order left と#pragma bit_order right で（データ並びイメージ）のように、それぞれ左右に割り付きます。

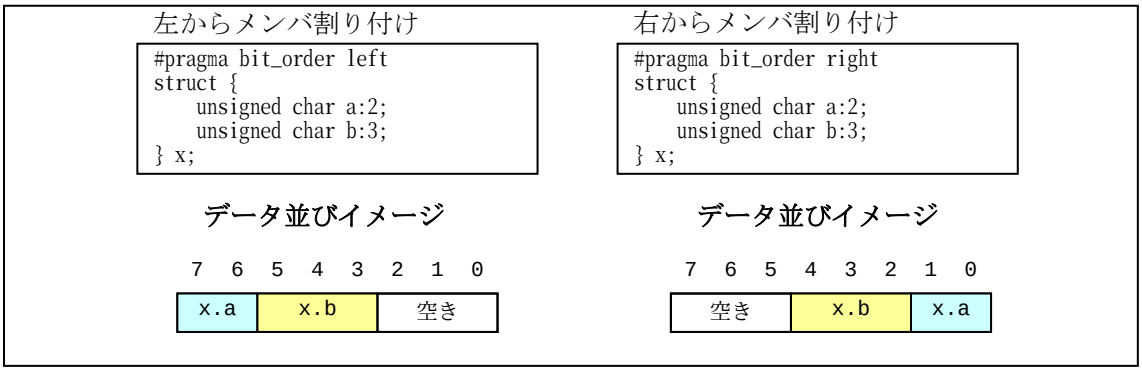


図 2-3

【例 2】

同じサイズの型指定子が連続している場合は、可能な限り同じ領域に詰め込みます。

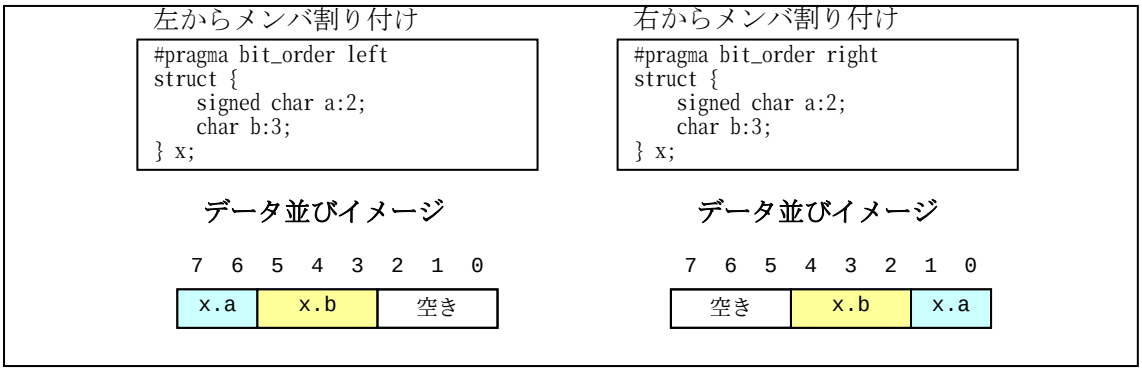


図 2-4

【例 3】

異なるサイズの型指定子が連続している場合は、次の領域に割り付けます。

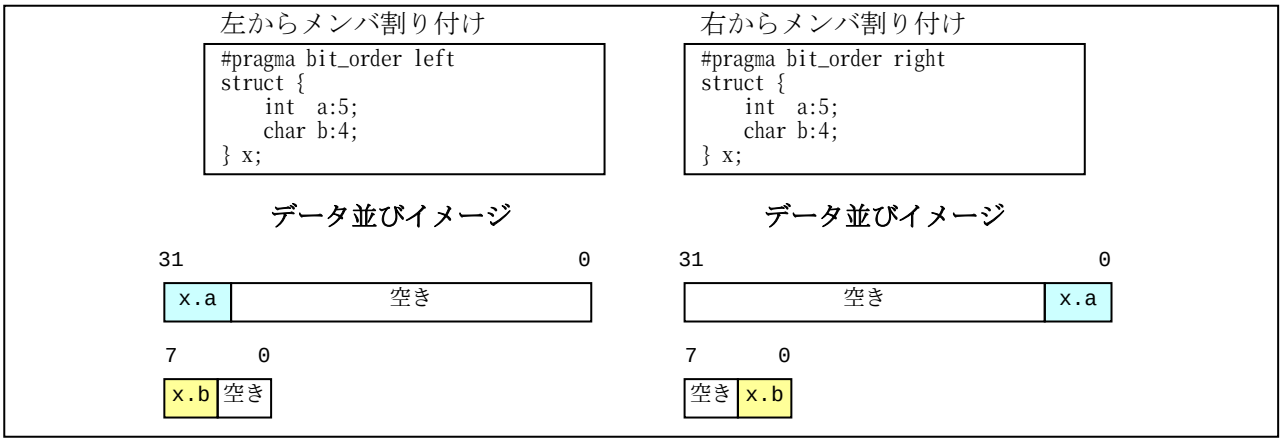


図 2-5

【例 4】

同じサイズの型指定子が連続していても、詰め込み先の領域の残りビットが、次のビットフィールドのサイズより小さい場合は、残りの領域は未使用領域となり、次の領域に割り付けます。

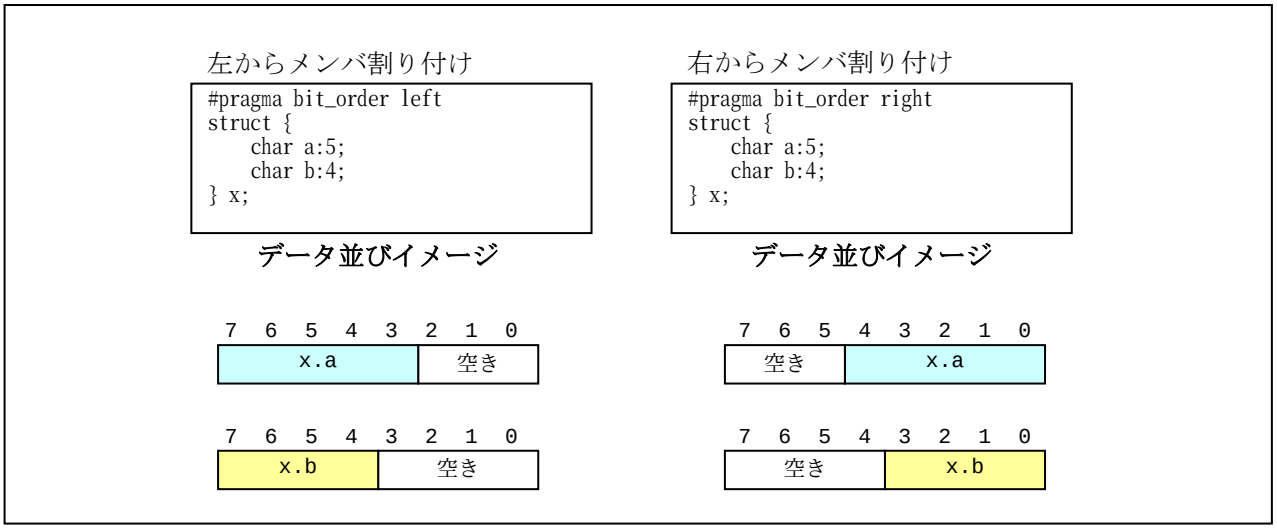


図 2-6

【例 5】

ビット幅 0 のビットフィールドのメンバを指定すると、次のメンバからは、強制的に次の領域に割り付けます。

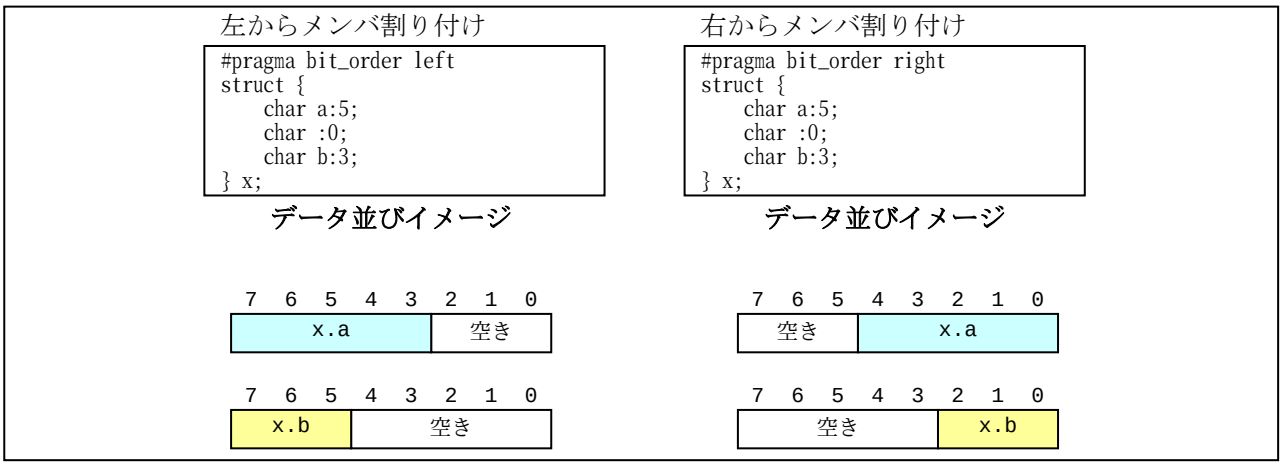


図 2-7

【例 6】

エンディアンのデフォルトはリトルエンディアンです。エンディアンはオプションで変更できます。リトルエンディアンを指定した場合(endian=little)、各領域の並び順はビックエンディアンでのバイトの並びと逆順になります。

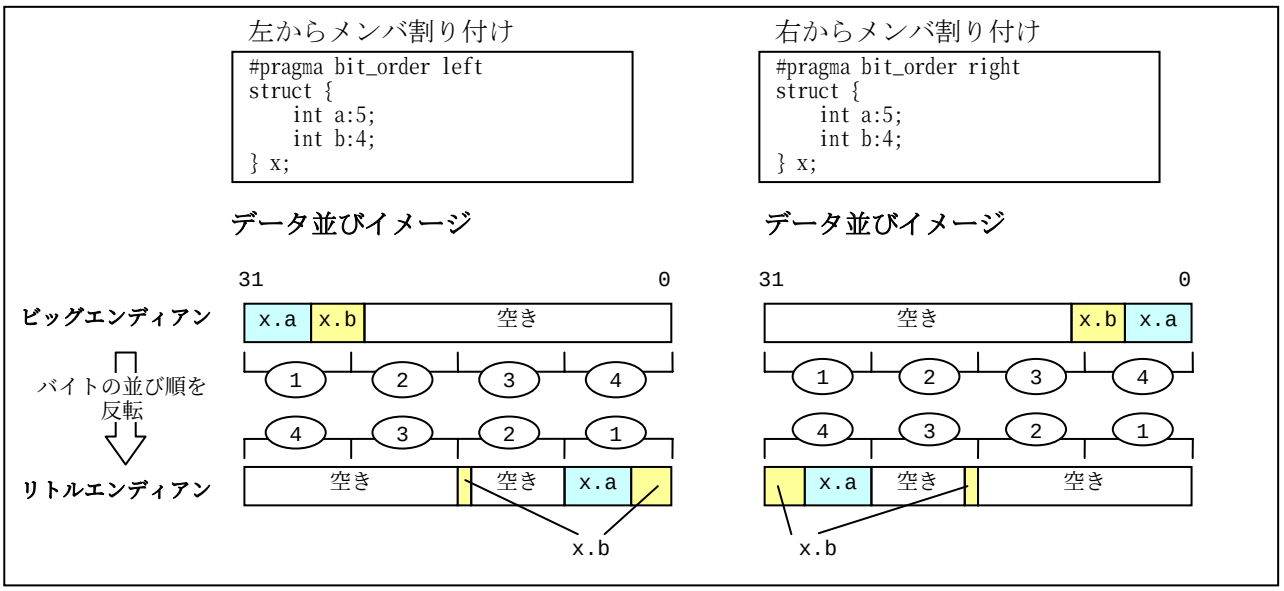


図 2-8

2.3 構造体メンバに対するアライメント制御

構造体において(共用体・クラスも同様です)、1 バイト、2 バイト、4 バイトのメンバが混在している場合、各メンバはそれぞれのアライメント数に従うためにメンバとメンバの間に空き領域が発生する場合があります。

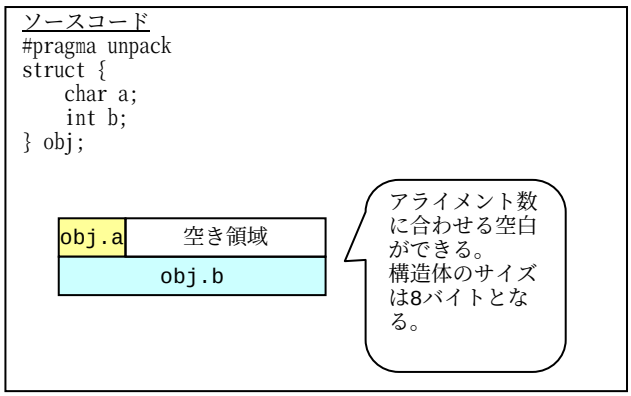


図 2-9

通信のプログラムで使用される構造体など、構造体に空き領域を作りたくない場合があります。この様な場合は、#pragma pack 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。メンバのアライメント数を 1 とした構造体には、メンバ間の空き領域が作られなくなります。ただし、アライメント数を 1 とした構造体メンバへアクセスする場合、アライメント数を変更する前と比較して、速度性能が低下する恐れがあります。

“pack” オプションを使用することで、全ての構造体メンバのアライメント数が 1 になります。“pack” オプションと #pragma が同時に指定された場合は、#pragma の指定が優先されます。

【書式】

#pragma pack
#pragma unpack
#pragma packoption

これらの指定によるアライメント数は次のようになります。

表 2-4 構造体、共用体、クラスメンバのアライメント数

指定	#pragma pack	#pragma unpack	#pragma packoption **または指定なし**
[signed]char	1	1	1
[unsigned]short	1	2	pack **オプションに従う**
[unsigned]int, [unsigned]long, [unsigned]long long, **浮動小数点数型**, **ポインタ型**	1	4	pack **オプションに従う**
アライメント数が 1 **の構造体、共用体、クラス**	1	1	1
アライメント数が 2 **の構造体、共用体、クラス**	1	2	pack **オプションに従う**
アライメント数が 4 **の構造体、共用体、クラス**	1	4	pack **オプションに従う**

【例】

構造体の割り付き方はそれぞれ以下のようになります。

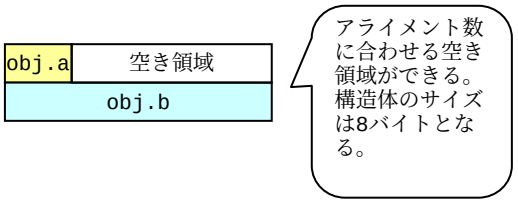
#pragma unpack 指定ソースコード

```
#pragma unpack
struct {
    char a;
    int b;
} obj;

void func(void)
{
    obj.b = 1;
    obj.a = 1;
}
```

アセンブラ展開コード

```
_func:                              ; function: func
            .STACK      _func=4
L10:
            MOV.L       #00000001H,R5
            MOV.L       #_obj,R4
            MOV.L       R5,04H[R4]
            MOV.B       R5,[R4]
            RTS
            .SECTION    B,DATA,ALIGN=4
            .glb        _obj
_obj:                               ; static: obj
            .blkl       2
```



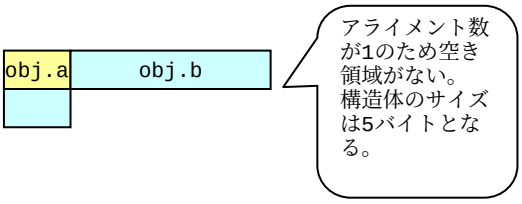
#pragma pack 指定ソースコード

```
#pragma pack
struct {
    char a;
    int b;
} obj;

void func(void)
{
    obj.b = 1;
    obj.a = 1;
}
```

アセンブラ展開コード

```
_func:                              ; function: func
            .STACK      _func=4
L10:
            MOV.L       #_obj,R3
            MOV.L       #00000001H,R5
            ADD         #01H,R3,R4
            MOV.L       R5,[R4]
            MOV.B       R5,[R3]
            RTS
            .SECTION    B_1,DATA
            .glb        _obj
_obj:                               ; static: obj
            .blkb       5
```



ホームページとサポート窓口<website and support>

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	2009.10.1	—	初版発行