RL78開発環境移行ガイド

R8CおよびM16CからRL78への移行
（コンパイラ編）
（High-performance Embedded Workshop, NC30WA→CS+,CC-RL）

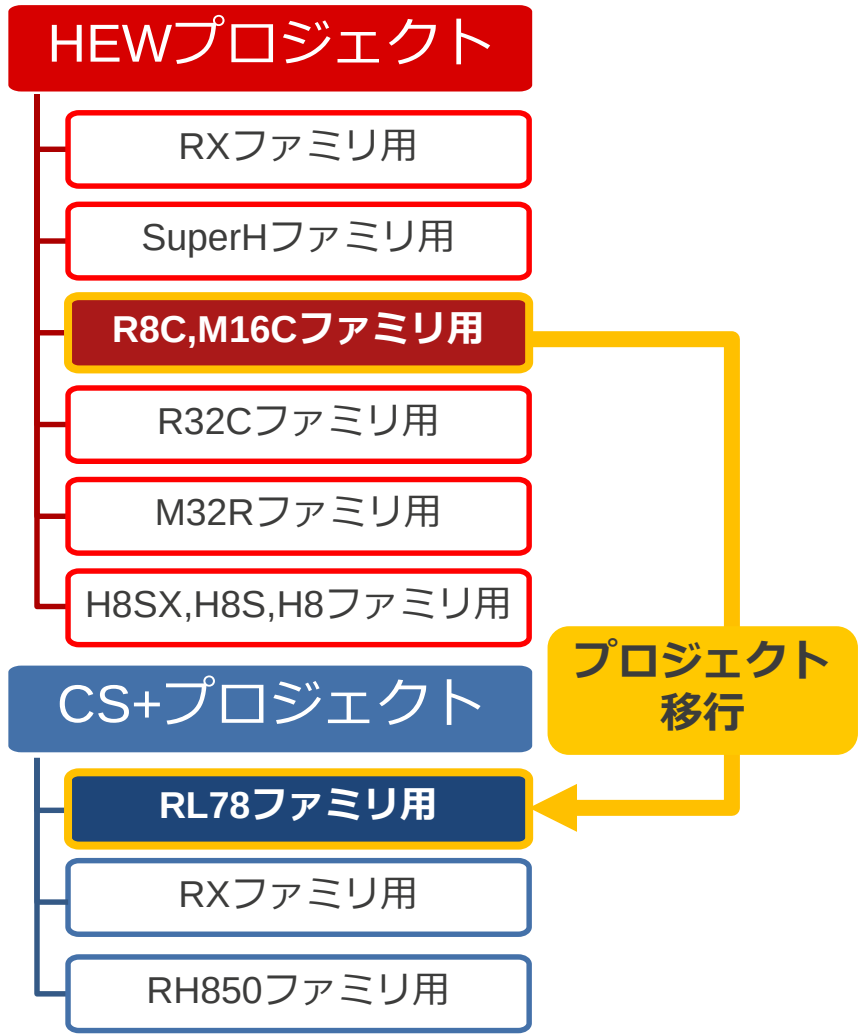
2016/12/28
R20UT2088JJ0300

ソフトウェア事業部ソフトウエア技術部
ルネサス システムデザイン株式会社

はじめに

- 本資料は、統合開発環境High-performance Embedded Workshop（以降、HEW）を用いて、R8C, M16Cファミリ用に作成したソースコードをRL78ファミリ用に移行する際のコーディング、オプションの差異について、 RL78ファミリ用でサポートしている内容を中心に記載しています。

- 本資料は、R8C, M16Cファミリ用 C/C++コンパイラパッケージ (M3T-NC30WA)、CS+、およびCC-RLをベースに説明しています。なお、対象のバージョンは以下の通りです。
  - R8C, M16Cファミリ用 C/C++コンパイラパッケージ V.6.00
  - CS+ for CC V4.01.00
  - CC-RL V1.03.00



アジェンダ

1.言語仕様

1-1.言語仕様の主な相違点(1/2)

		R8C/M16C	RL78	説明項番	説明ページ	対応方法
型	double/long double	64ビット	32ビット	1-2(a)	P7	オプション対応可
修飾子	near/far	near/far _near/_far	__near/__far	1-2(a)	P8	ソース変更
	restrict	サポート	未サポート	1-2(a)	P9	ソース変更
	_ext4mptr	_ext4mptr far	__far	1-2(a)	P9	ソース変更
	inline/_inline	inline/_inline	__inline	1-2(a)	P10	ソース変更

1-1.言語仕様の主な相違点(2/2)

		R8C/M16C	RL78	説明項番	説明ページ	対応方法
拡張機能	asm関数	asm(“ “); _asm(“ “);	#pragma inline_asm	1-2(b)	P11	ソース変更
	asm記述	#pragma ASM ～ #pragma ENDASM	#pragma inline_asm	1-2(b)	P11	ソース変更
	セクション名変更	#pragma SECTION	#pragma section	1-2(b)	P12	ソース変更
	割り込み宣言	#pragma INTERRUPT	#pragma interrupt	1-2(b)	P15	ソース変更
	構造体配置	#pragma STRUCT	-pack	1-2(b)	P16	オプション対応可
	スペシャルページ	#pragma SPECIAL	__callt #pragma callt	1-2(b)	P17	ソース変更
	汎整数拡張	拡張しない(オプション制御可)	常に拡張する	-----	-----	ソース変更

1-2. R8C/M16CからRL78へ移植する場合のポイント
(a)標準C言語(C89,C99)仕様、実装依存の相違による対処方法(1/4)

- double/long double型の対処方法

  -dbl_sizeオプションを指定してください。

R8C/M16C	オプション	
	指定なし	-fdouble_32
型のサイズ	64ビット	32ビット

RL78	オプション		
	指定なし	-dbl_size=4	-dbl_size=8
型のサイズ	32ビット	32ビット	64ビット

1-2. R8C/M16CからRL78へ移植する場合のポイント
(a)標準C言語(C89,C99)仕様、実装依存の相違による対処方法(2/4)

- near/far修飾子の対処方法
  - R8C/M16Cの表現

    near,far,_near,_far
  - RL78の表現

    __near,__far

  R8C/M16Cの表現をRL78の表現に置換する#defineを記述してください。

  #define near __near

  なお、-convert_cc=nc30を指定してる場合は対応不要です。

1-2. R8C/M16CからRL78へ移植する場合のポイント
(a)標準C言語(C89,C99)仕様、実装依存の相違による対処方法(3/4)

- restrict修飾子の対処方法

  #defineで無効化してください。

  ```
  #define restrict
  ```

  なお、-convert_cc=nc30を指定してる場合は対応不要です。

- _ext4mptr修飾子の対処方法

  #defineで無効化してください。

  ```
  #define _ext4mptr
  ```

  _ext4mptr修飾子で宣言した変数は、farポインタとして、far宣言しています。

  合わせて、far修飾子を__far修飾子 に#defineで置換してください。

  なお、-convert_cc=nc30を指定してる場合は対応不要です。

1-2. R8C/M16CからRL78へ移植する場合のポイント
(a)標準C言語(C89,C99)仕様、実装依存の相違による対処方法(4/4)

- inline/_inline修飾子の対処方法

  inline/_inline修飾子を__inline修飾子に#defineで置換してください。

  ```
  #define inline __inline
  ```

  なお、-convert_cc=nc30を指定してる場合は対応不要です。

1-2. R8C/M16CからRL78へ移植する場合のポイント
(b)拡張機能の相違による対処方法(1/7)

- asm関数の対処方法

  asm関数の引数の命令を記述した関数定義を用意して#pragma inline_asmを指定してください。

  asm関数の引数は削除して、関数呼び出しと同様にしてください。

```
#pragma inline_asm asm
void asm(void) {
  RL78の命令
}
```

- #pragma ASM～#pragma ENDASMの対処方法

  #pragma ASM～#pragma ENDASM間の命令を記述した関数定義を用意して

  #pragma inline_asm を指定してください。

  その関数を#pragma ASM～#pragma ENDASMの位置で呼び出してください。

1-2. R8C/M16CからRL78へ移植する場合のポイント
(b)拡張機能の相違による対処方法(2/7)

- #pragma SECTIONの対処方法
  - R8C/M16Cコンパイラの書式

    #pragma SECTION 規定セクション名 変更セクション名
  - RL78コンパイラの書式

    #pragma section セクション種別 変更セクション名

  次ページに規定セクション名とセクション種別の相違を記載します。

  参考にして規定セクション名を書き換えてください。

  なお、-convert_cc=nc30を指定してる場合は

  無効な書式に対して#pragma 指令を削除し警告メッセージを出力します。

1-2. R8C/M16CからRL78へ移植する場合のポイント
(b)拡張機能の相違による対処方法(3/7)

- #pragma SECTIONの対処方法(2)

  規定セクション名とセクション種別は、以下の表に従い変更してください。

	R8C/M16C	RL78
プログラム	program	text
const変数	rom	const
初期値あり変数	data	data
初期値なし変数	bss	bss

1-2. R8C/M16CからRL78へ移植する場合のポイント
(b)拡張機能の相違による対処方法(4/7)

- #pragma SECTIONの対処方法(3)

  変更後のセクション名に以下の表に従い配置属性が付加されます。

	R8C/M16C	RL78
near	_N	_n
far	_F	_f
SBDATA	_S	未サポート
saddr	未サポート	_s

また、R8C/M16Cでは、データサイズの属性(E/O)とデータの初期値を保持するセクションの属性(I)を付加しますが、RL78には該当する属性はサポートしていません。

そのため、変更後のセクション名に従い、リンク時のセクションの配置指示を書き換えてください。

1-2. R8C/M16CからRL78へ移植する場合のポイント
(b)拡張機能の相違による対処方法(5/7)

- #pragma INTERRUPTの対処方法
  - R8C/M16Cコンパイラの書式

    #pragma INTERRUPT△[/B¦/E]△割り込み処理関数名

    #pragma INTERRUPT△[/B¦/E]△割り込みベクタ番号△割り込み処理関数名

    #pragma INTERRUPT△[/B¦/E]△割り込み処理関数名(vect=割り込みベクタ番号)
  - RL78コンパイラの書式

    #pragma interrupt [(] 割り込みハンドラ名[(割り込み仕様 [,...])][)]

  上記フォーマットに従って書き換えてください。

  なお、-convert_cc=nc30を指定してる場合は

  無効な書式に対して#pragma 指令を削除し警告メッセージを出力します。

1-2. R8C/M16CからRL78へ移植する場合のポイント
(b)拡張機能の相違による対処方法(6/7)

- #pragma STRUCTの対処方法

  RL78では#pragma STRUCTに該当する書式はサポートしていません。

  -packオプションでファイル毎のpack指定が可能です。(タグ毎の指定はできません。)

  #pragma STRUCTを削除して、以下の通りにオプションを指定してください。

  また、メンバを並び替えるarrangeは未サポートです。

  - RL78コンパイラのオプション

    -pack指定ありの場合、packします。

    -pack指定なしの場合、packしません。(unpackと同等です。)

  なお、-convert_cc=nc30を指定してる場合は

  #pragma 指令を削除し警告メッセージを出力します。

1-2. R8C/M16CからRL78へ移植する場合のポイント
(b)拡張機能の相違による対処方法(7/7)

- #pragma SPECIALの対処方法
  - R8C/M16Cコンパイラの書式

    #pragma SPECIAL△呼び出し番号△関数名()

    #pragma SPECIAL△関数名(vect=呼び出し番号)
  - RL78コンパイラの書式

    __callt 関数名();

    #pragma callt 関数名

上記フォーマットに従って書き換えてください。なお、-convert_cc=nc30を指定してる場合は対応不要です。

✓ 注意点

スペシャルページは、238個宣言可能ですが、callt関数は 32個になります。utl30を使用したスペシャルページでは、参照回数の多い関数をベクタ番号255から割り当てています。参考にしてください。

2.オプション

2-1. オプションの主な相違点および共通点(1/3)

オプションカテゴリ	種別	R8C/M16C	RL78	効果の相違	項番	ページ
コンパイル結果ファイル出力	オブジェクト名変更	-o ファイル名	-o ファイル名		-------	
	アセンブラファイルのみの出力	-S	-S		-------	
	Cソース付アセンブラリスト出力	-dS	-pass_source	-Sオプションと合わせて指定する	-------	
	プリプロセス処理結果のみ出力	-P	-P		-------	
最適化	ROMサイズ優先	-OR	-Osize		2-2(b)	P23
	実行速度優先	-OS	-Ospeed		2-2(c)	P24
	標準の最適化	-O	-Odefault		2-2(a)	P22
	分岐最適化	-OGJ ←V5 -goptimize ←V6	-	リンカの-optimize=branchオプションを指定する	2-2(d)	P25

2-1. オプションの主な相違点および共通点(2/3)

オプション カテゴリ	種別	R8C/M16C	RL78	効果の相違	項番	ページ
プリプロセス	マクロ定義	-Dxxx	-Dxxx	RL78は複数のマクロを定義する場合は”,”に続けて記述可能	----	----
	マクロ定義を無効	-Uxxxx	-Uxxxx	RL78は複数のマクロ定義を無効にする場合は”,”に続けて記述可能。 R8Cは無効にできるのはプリデファインドマクロのみ	----	----
	インクルードパス指定	-I	-I	RL78は複数のパスを指定する場合は”,”に続けて記述可能	----	----
クロスリファレンス情報		xrf30で出力　←V5 optlnkで対応　←V6	-cref		----	----

2-1. オプションの主な相違点および共通点(3/3)

オプションカテゴリ	種別	R8C/M16C	RL78	効果の相違	項番	ページ
最適化	共通コードの関数化	-O R 指定時	-Osame_code=on		----	----
	共通コードの関数化抑止	-Ono_asmopt(-OA)	-Osame_code=off		----	----
	デバッグ情報優先	-Ono_break_source_debug (-ONBSD)　*他の最適化併用要	-Onothing		2-2(d)	P26
	レジスタ割付	-fenable_register(-fER)	-	-Ospeed/-Odefault/-Osizeオプションを指定して、最適化を有効にする	----	----
	switchテーブル化	デフォルト	-switch=rel_table		----	----
	switchテーブル化抑止	-fno_switch_table(-fNST)	デフォルト		----	----
デバッグ情報		-g or –finfo	-g		----	----
警告		-Wall等	-	警告メッセージは常に出力する	----	----

2-2. R8C/M16Cコンパイラから RL78コンパイラへ変更する場合のポイント

(a) 標準的な最適化オプション使用時のガイド

R8C/M16Cコンパイラ：-O[1-5]を使用していた場合、

➡ RL78では、-Odefaultを選択します。

◆有効になる最適化

- ●未使用static関数の削除
- ●パイプライン最適化
- ●関数末尾の関数呼び出しのbr命令置き換え

2-2. R8C/M16Cコンパイラから RL78コンパイラへ変更する場合のポイント

(b) コードサイズ優先オプション使用時のガイド

R8C/M16Cコンパイラ：-ORを使用していた場合、

➡ RL78では、-Osizeを選択します。

◆有効になる最適化

- ●未使用static関数の削除
- ●パイプライン最適化
- ●関数末尾の関数呼び出しのbr命令置き換え
- ●共通コードの関数化

2-2. R8C/M16Cコンパイラから RL78コンパイラへ変更する場合のポイント

(c) スピード優先オプション使用時のガイド

R8C/M16Cコンパイラ：-OSを使用していた場合、

➡ RL78では、-Ospeedを選択します。

◆有効になる最適化

- ●未使用static関数の削除
- ●パイプライン最適化
- ●関数末尾の関数呼び出しのbr命令置き換え
- ●関数のインライン展開

2-2. R8C/M16Cコンパイラから RL78コンパイラへ変更する場合のポイント

(d) その他の最適化オプション使用時のガイド(1/2)

R8C/M16Cコンパイラで分岐最適化(V5:-Oglobal_jmp(-OGJ),V6:-goptimize）を使用している場合、

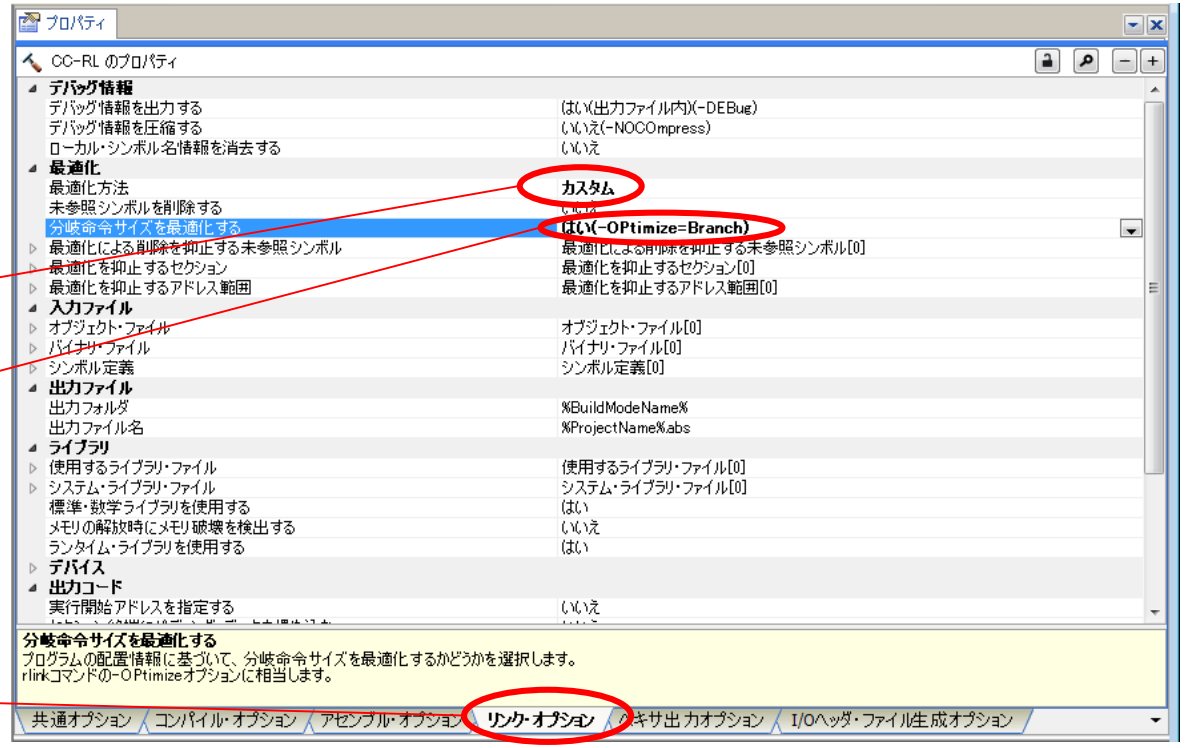
RL78では、CS+でリンク・オプションの「分岐命令サイズを最適化する」を選択します。

①リンク・オプションタブを選択

②最適化方法：カスタム を選択

③分岐命令サイズを最適化する

：はい（-Optimize=Branch)を選択



2-2. R8C/M16Cコンパイラから RL78コンパイラへ変更する場合のポイント

(d) その他の最適化オプション使用時のガイド(2/2)

R8C/M16Cコンパイラでデバッグ情報を優先する最適化(-Ono_break_source_debug(-ONBSD)を使用している場合、

➡ RL78では、CS+でコンパイル・オプションの「最適化レベル：デバッグ優先」を選択します。

①コンパイル・オプションタブを選択

②最適化レベル：デバッグ優先
(-Onothing) を選択

3.コード削減

3-1. 拡張機能でオブジェクト生成の効率化
(a) saddr 領域、callt 領域の利用(1/2)

RL78 C コンパイラではデバイスのsaddr 領域、callt 領域を利用することにより、効率の良いオブジェクトを生成することができます。

■ 外部変数を使用する

外部変数を定義するときにsaddr領域が利用可能であれば、定義する外部変数に __saddr宣言をします。__saddr宣言をした変数は、メモリに対する命令と比べ命令コードが短く、オブジェクト・コードを縮小することができ、実行速度も向上します。
__saddr宣言ではなく、#pragma saddrで指定することも可能です。

```
__saddr      型名 変数名 / __saddr static      型名 変数名
```

```
#pragma saddr      変数名
```

※変数/関数情報ファイルを利用することで自動で行うことも可能です。

3-1. 拡張機能でオブジェクト生成の効率化
(a) saddr 領域、callt 領域の利用(2/2)

■ 関数定義の工夫

何回も呼ばれる関数で、callt領域を利用できる場合はcallt関数にします。

callt関数は、デバイスのcallt領域を利用して呼び出されるので、通常の呼び出しよりも短いコードで呼び出すことができます。

```
__callt  extern  型名 関数名
```

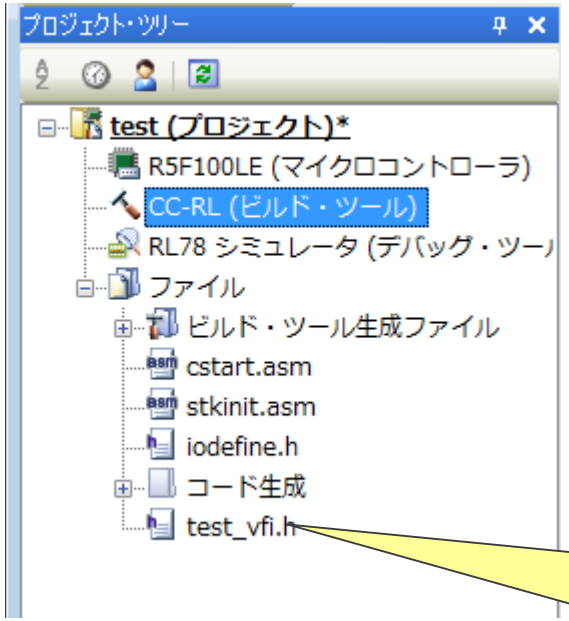
```
#pragma callt  関数名
```

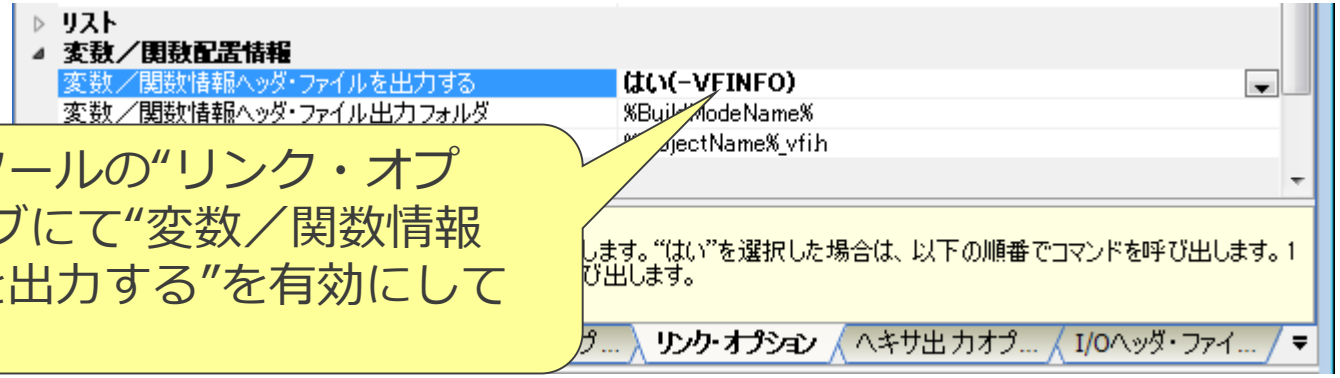
※変数/関数情報ファイルを利用することで自動で行うことも可能です。

3-1. 拡張機能でオブジェクト生成の効率化
(b)変数/ 関数情報ファイルの利用

変数/関数情報ファイルを利用することで、変数や関数を効率よく配置することができます。

参照頻度の高い変数はsaddr領域へ、参照頻度の高い関数はcallt関数として扱います。





ビルド・ツールの“リンク・オプション”タブにて“変数／関数情報ファイルを出力する”を有効にしてください。

参照頻度の高い変数、関数の情報ファイルをプロジェクト・ツリーに登録されます。

3-1. 拡張機能でオブジェクト生成の効率化
(c) ミラー領域の利用(1/4)

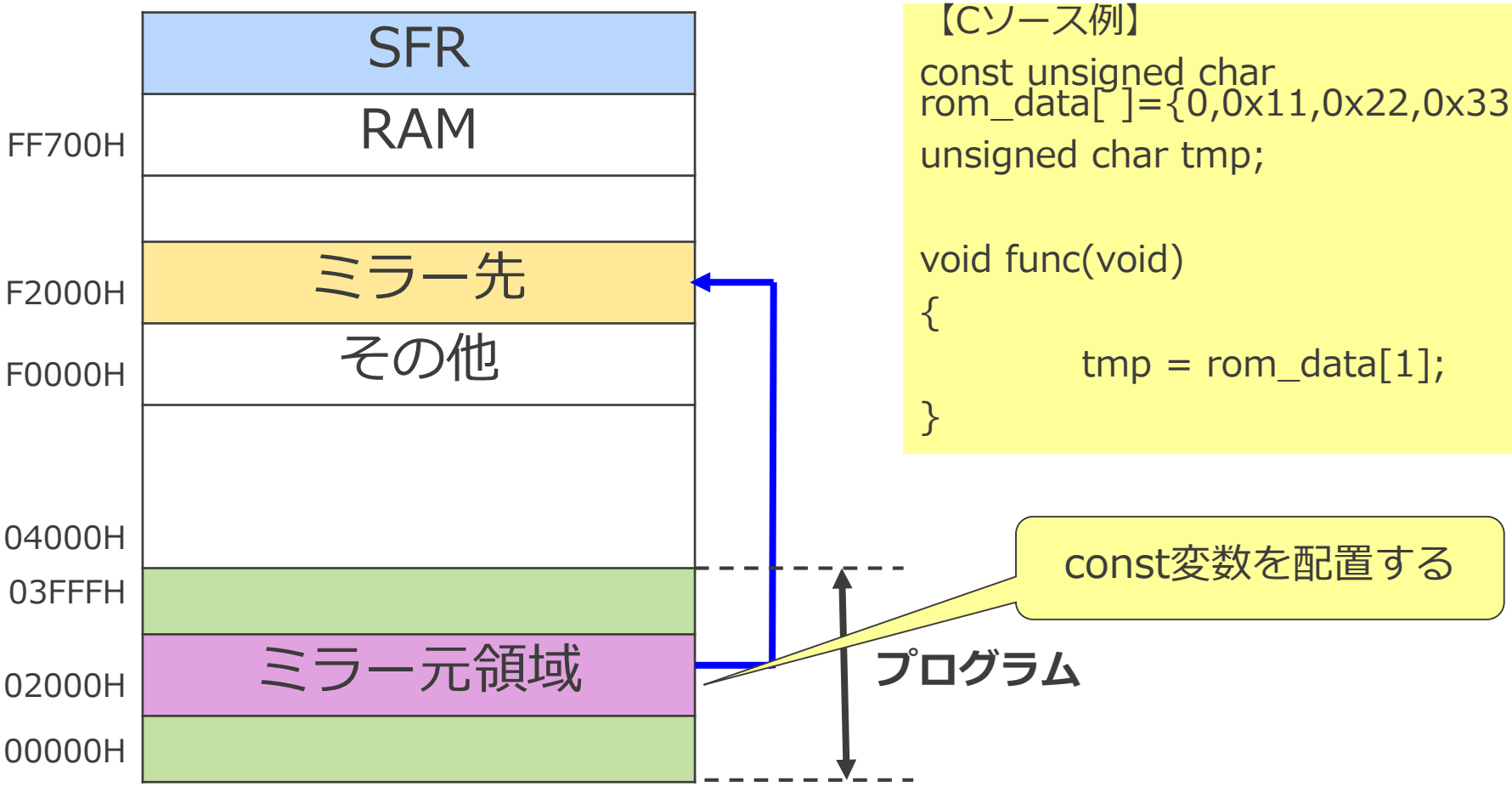
const変数などのROMデータはミラー領域から読み出すことで、オペランドにESレジスタを持たない命令を使用することができるため、短いコードでコード・フラッシュ内容の読み出しを行うことができます。

```
const unsigned char rom_data[ ] = {0x11, 0x22, 0x33, 0x44};
unsigned char tmp;

void func()
{
    tmp = rom_data[1];
 }
```

3-1. 拡張機能でオブジェクト生成の効率化
(c) ミラー領域の利用(2/4)

ROMデータをミラー領域(near扱い)に配置したイメージ図です。



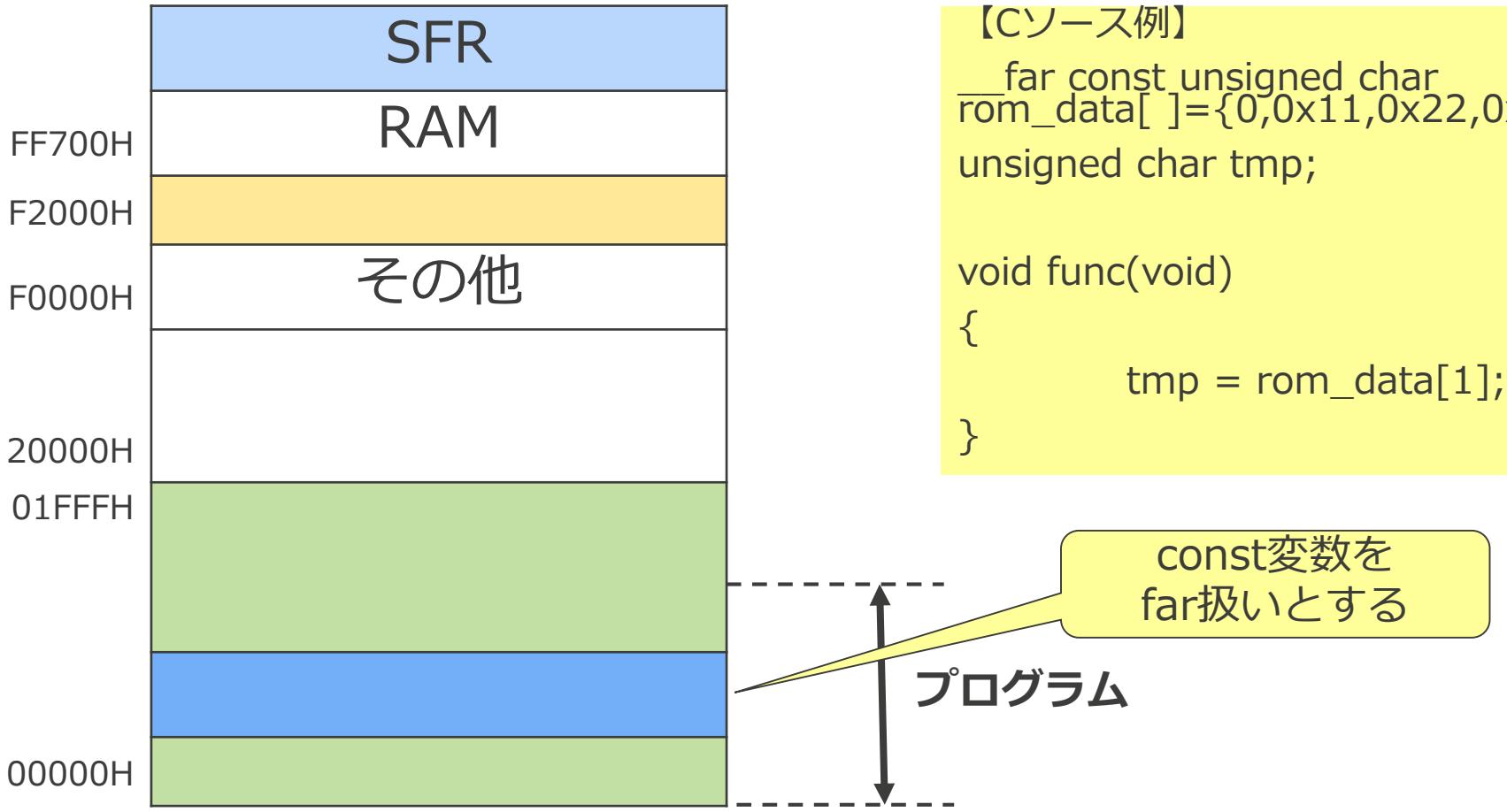
【Cソース例】

```
const unsigned char
rom_data[ ]={0,0x11,0x22,0x33};
unsigned char tmp;

void func(void)
{
        tmp = rom_data[1];
}
```

3-1. 拡張機能でオブジェクト生成の効率化
(c) ミラー領域の利用(3/4)

ROMデータをミラー領域外(far扱い)に配置したイメージ図です。



【Cソース例】

```
__far const unsigned char
rom_data[ ]={0,0x11,0x22,0x33};
unsigned char tmp;

void func(void)
{
        tmp = rom_data[1];
}
```

3-1. 拡張機能でオブジェクト生成の効率化
(c) ミラー領域の利用(4/4)

ミラー領域に配置した方がコードが短くなります。

【ミラー領域からのアクセスするアセンブラ・ソース例】

```
;***       6 :            tmp = rom_data[1];
mov !LOWW(_tmp), #0x11
ret
```

【ミラー領域以外からのアクセスするアセンブラ・ソース例】

```
;***       6 :            tmp = rom_data[1];
mov es, #LOW(HIGHW(_rom_data))
mov a, es:!LOWW(_rom_data+0x00001)
mov !LOWW(_tmp), a
ret
```

3バイトの増加

ルネサス システムデザイン株式会社