RH850の割り込み/例外実現方法

CC-RH アプリケーションガイド

R20UT3546JJ0101

2018.10.12

ソフトウェア開発統括部、ソフトウェア技術部
ルネサス エレクトロニクス株式会社

アジェンダ

概要

RH850の割り込み/例外方法について

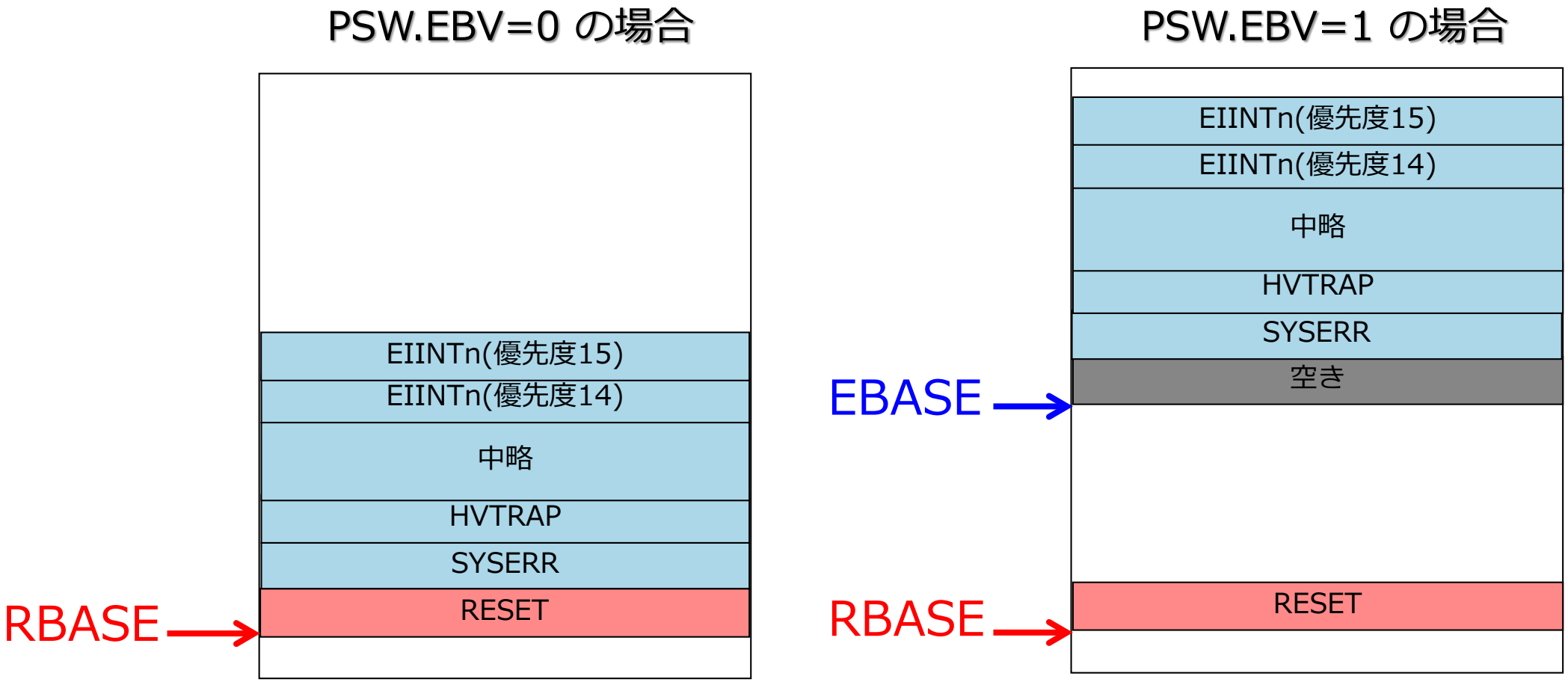
RH850には直接ベクタ方式とテーブル参照方式の2種類の割り込み/例外があります。

直接ベクタ方式は、発生要因毎に固定のハンドラ・アドレスにジャンプして、ジャンプ先のコードを実行します。RBASEあるいはEBASEのいずれをベース・アドレスとして、発生要因のオフセット値を加算した値をハンドラ・アドレスとします。

テーブル参照方式は、ハンドラ・アドレスに格納されたワードデータを読み出して、そのワードデータが指すアドレスにジャンプします。
INTBPをベース・レジスタとして、チャネル番号*4のオフセット値を加算した値をハンドラ・アドレスとします。

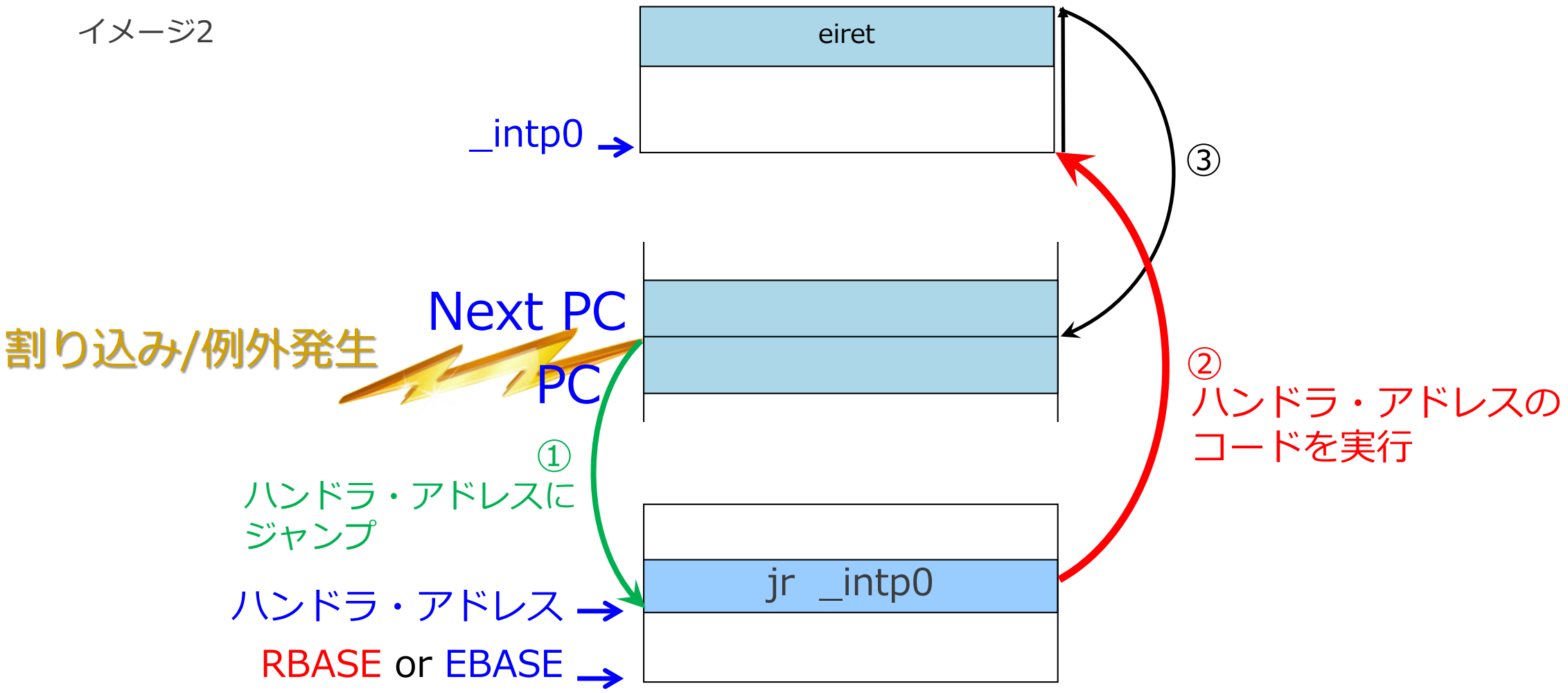
直接ベクタ方式

イメージ1



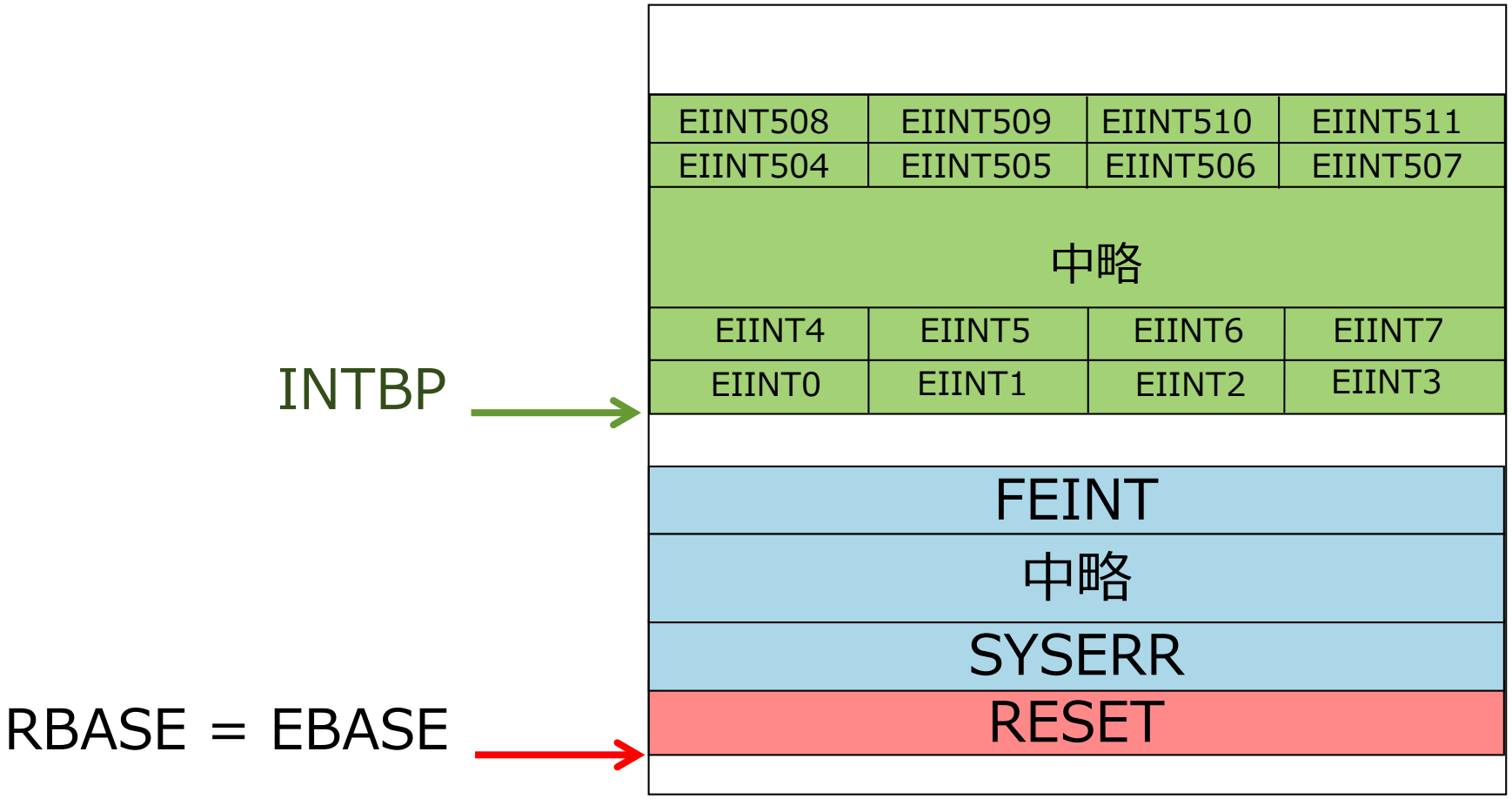
直接ベクタ方式

イメージ2



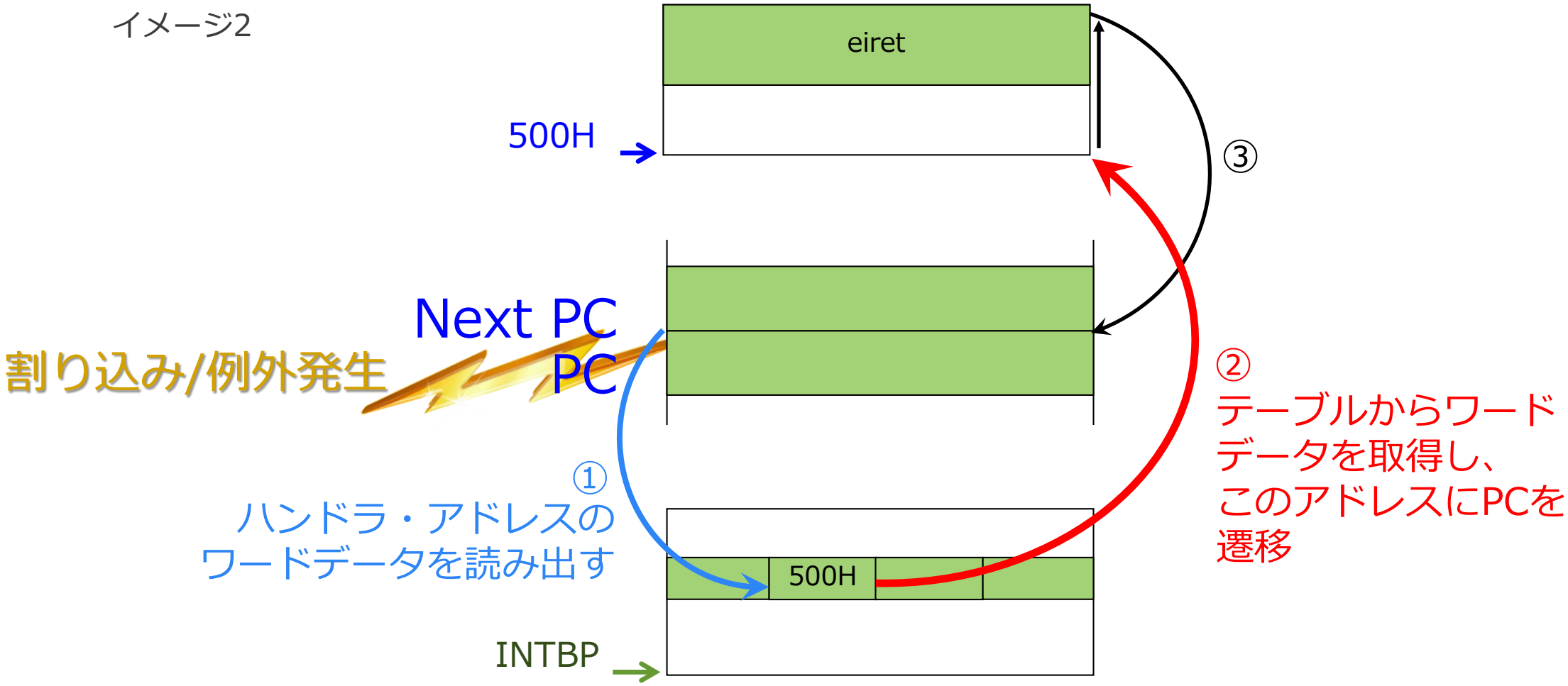
テーブル参照方式

イメージ1



テーブル参照方式

イメージ2



割り込み/例外の実現方法

CC-RHを使用して以下の2ステップにより、RH850の割り込み/例外を実現してください。

① #pragma interrupt 指令を用いた
　割り込み/例外関数の定義

② 割り込み/例外ベクタの定義

本アプリケーションノートでは、 CC-RHを使用して直接ベクタ方式/テーブル参照方式の割り込み/例外を実現する手法をそれぞれ説明します。

割り込み/例外発生時に実行する関数の定義

割り込み/例外関数の定義方法1

割り込み関数をC言語で記述する場合、#pragma interrupt 指令を使用してください。これにより、C言語で記述した関数を割り込み関数としてCC-RHがコンパイルします。

```
#pragma interrupt 関数名　(割り込み仕様 [,割り込み仕様])
```

関数名で指定した関数を割り込み仕様で指定した仕様に従ってコンパイルします。
通常関数とは関数本体を実行するまでの入口/出口コードが異なります。

割り込み関数の注意事項：

- ✓ 戻り型は常にvoidとしてください。
- ✓ 引数なし、または引数1個のみ使用できます。引数はEIIC(EIレベル例外要因)またはFEIC(FEレベル例外要因)です。
- ✓ 通常関数のように呼び出さないでください。

割り込み/例外関数の定義方法2

割り込み仕様	機能
enable=	多重割り込みの可否を指定します。 引数にはtrue/false/manual が記載できます。
priority=	直接ベクタ方式の割り込み発生要因を指定します。
channel=	テーブル参照方式の割り込みチャネル番号を指定します。priority/channel はどちらか一方のみ指定可能です。 両方指定した場合はエラーとなります。 （注）どのチャネル番号を指定しても、同じEIレベル割り込みとしてコードを出力します。
fpu=	FPU機能システムレジスタFPEPC/FPSR を退避/復帰するか指定します。 引数にはtrue/false/auto が記載できます。FPUを使用している場合にはtrueを指定してください。
fxu=	fxrs/fxxpを退避/復帰するか指定します。 引数にはtrue/false/auto が記載できます。 （注）-Xcpu=g4mh が指定されていない場合はエラーになります。
callt=	CTPC/CTPSW を退避/復帰するか指定します。 引数にはtrue/false が記載できます。 （注）コンパイル時にcallt命令を生成することはありません。そのためCTPC/CTPSWの退避/復帰は不要です。
resbank	resbank命令を関数の出口コードに出力します。 割り込み仕様priority=に、EIINT以外を同時に指定した場合はエラーになります。 （注）-Xcpu=g4mh が指定されていない場合はエラーになります。
param=	例外要因レジスタの値を引数で受け取る方法を指定します。 例外要因レジスタ名は1から4個まで指定できます。

割り込み/例外関数の定義方法3

#pragma interrupt 指令の割り込み仕様にpriorityあるいはchannel を指定することにより、以下のRH850割り込み/例外関数を定義することが可能です。

- EIレベル割り込み
- FEレベル割り込み（復帰/回復可）
- FEレベル割り込み（復帰/回復不可）

以降のページでは、それぞれの割り込み/例外関数の出力コードの内容を説明します。

EIレベル割り込み関数の出力コード

#pragma interrupt 指令の割り込み仕様としてpriority=EIINT/FPI等、あるいはchannelを指定すると、EIレベル割り込み関数として右記のような入口コード(1.～7.)と出口コード(8.～15.)を出力します。

1. 使用するスタック領域を確保
2. 割り込み関数中で使用するCaller-Saveレジスタを退避
3. EIPC，EIPSW を退避　(enable=true|manua l 指定時)
4. 仮引数を記述した関数の場合、EIIC をR6 に設定
5. 多重割り込みを許可　(enable=true 指定時)
6. CTPC，CTPSW を退避　(callt=true 指定時)
7. FPEPC，FPSR を退避　(fpu=true 指定時)

割り込み関数としてユーザーが記述した処理

8. インプレサイス割り込み待ちを設定
9. FPEPC，FPSR を復帰　(fpu=true 指定時)
10. CTPC，CTPSW を復帰　(callt=true 指定時)
11. 多重割り込みを禁止　(enable=true 指定時)
12. EIPC，EIPSW を復帰　(enable=true|manua l 指定時)
13. 割り込み関数中で使用したCaller-Saveレジスタを復帰
14. 使用したスタック領域を解放
15. eiret

FEレベル割り込み関数の出力コード （復帰/回復可）

#pragma interrupt 指令の割り込み仕様としてpriority=FEINT/PIE等のような復帰/回復可能な割り込み発生要因を指定すると、FEレベル割り込み関数として右記のような入口コード(1.～5.)と出口コード(6.～10.)を出力します。

1. 使用するスタック領域を確保
2. 割り込み関数中で使用するCaller-Saveレジスタを退避
3. 仮引数を記述した関数の場合、FEIC をR6 に設定
4. CTPC，CTPSW を退避 (callt=true 指定時)
5. FPEPC，FPSR を退避 (fpu=true 指定時)

割り込み関数としてユーザーが記述した処理

6. FPEPC，FPSR を復帰 (fpu=true 指定時)
7. CTPC，CTPSW を復帰 (callt=true 指定時)
8. 割り込み関数中で使用したCaller-Saveレジスタを復帰
9. 使用したスタック領域を解放
10. feret

FEレベル割り込み関数の出力コード （復帰/回復不可）

#pragma interrupt 指令の割り込み仕様としてpriority=FENMI/SYSERR等のような復帰/回復不能な割り込み発生要因を指定すると、FEレベル割り込み関数として右記のような入口コード(1.)を出力します。

出口コードは出力しないため、abort() を呼び出してプログラムを終了させる等、ユーザープログラム内で適切に処置してください。

1. 仮引数を記述した関数の場合、FEIC をR6 に設定
   割り込み関数としてユーザーが記述した処理

なし

直接ベクタ方式のベクタを定義

ハンドラ・アドレスの決定方法

直接ベクタ方式のハンドラ・アドレスは(RESETを除く)
PSW.EBV=0の場合にはRBASE＋オフセット・アドレス、
PSW.EBV=1の場合にはEBASE＋オフセット・アドレスとなります。

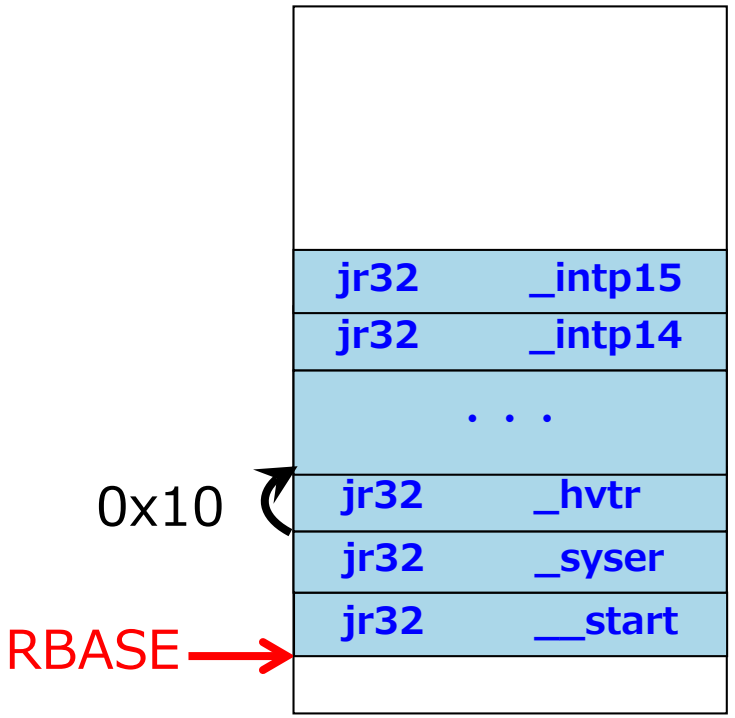
割り込み/例外要因	PSW.EBV=0	PSW.EBV=1	オフセット・アドレス
	ベース・レジスタ		
RESET	RBASE	なし	000H
SYSERR		EBASE	010H
HVTRAP			020H
FETRAP			030H
TRAP0			040H
TRAP1			050H
RIE			060H
FPP/FPI			070H
UCPOP			080H

直接ベクタ方式のベクタ定義1

ベクタは各要因ごとに0x10バイト確保されています。アセンブラで各要因ごとに実行したい関数へのジャンプ命令を記述することでベクタを定義してください。

例：ベース・レジスタをRBASE固定で使用する場合、
　またはRBASE=EBASEで使用する場合

```
.section "RESET", text
jr32      __start          ; RESET
.align    0x10
jr32      _syser           ; SYSERR
.align    0x10
jr32      _hvtr            ; HVTRAP
 ・・・
.align    0x10
jr32      _intp14          ; EIINTn(優先度14)
.align    0x10
jr32      _intp15          ; EIINTn(優先度15)
```



直接ベクタ方式のベクタ定義2

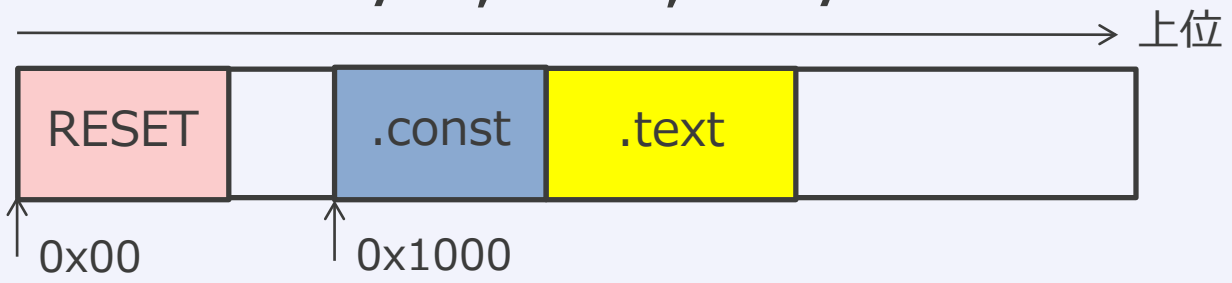
RESETセクションをリセット・ベクタ・アドレス（RBASEの値）に配置させてください。

セクションの配置アドレスはリンカオプション”-start”で指定します。

```
-start=section1,section2,.../address
```

例：0x00番地から上位方向にRESETセクションを、0x1000番地から
上位方向に.const、.textセクションを配置させる場合

-start=RESET/00,.const,.text/1000

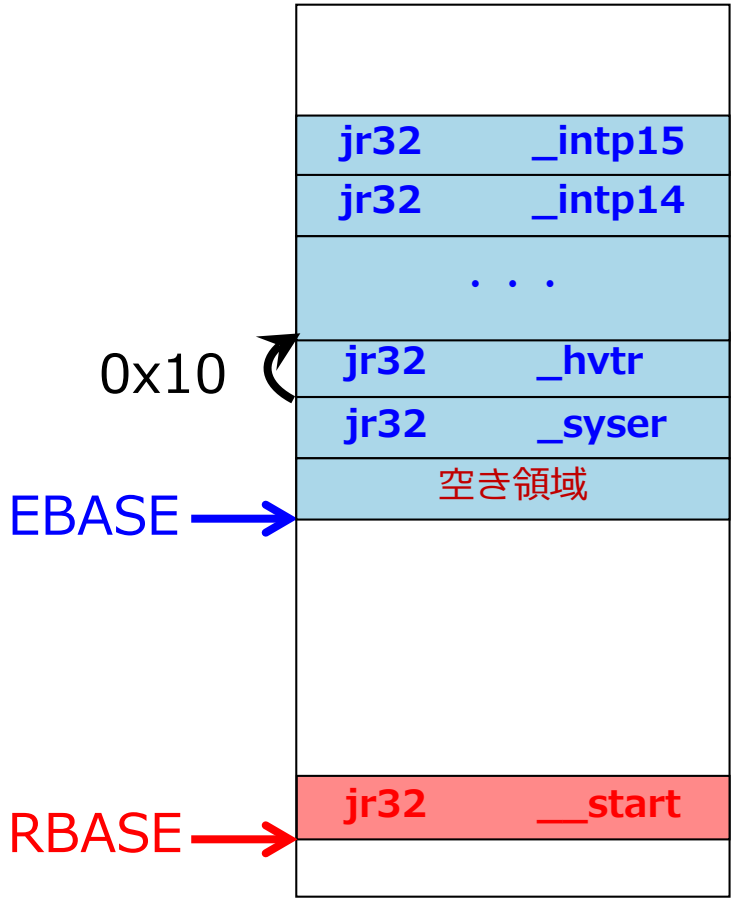


直接ベクタ方式のベクタ定義3

例：RBASE≠EBASEで使用する場合

```
.section "RESET", text
jr32      __start              ; RESET
```

```
.section “VECT", text
.ds       0x10                 ; 空き領域
jr32      _syser               ; SYSERR
.align    0x10
jr32      _hvtr                ; HVTRAP
 ・・・
.align    0x10
jr32      _intp14              ; EIINTn(優先度14)
.align    0x10
jr32      _intp15              ; EIINTn(優先度15)
```



直接ベクタ方式のベクタ定義4

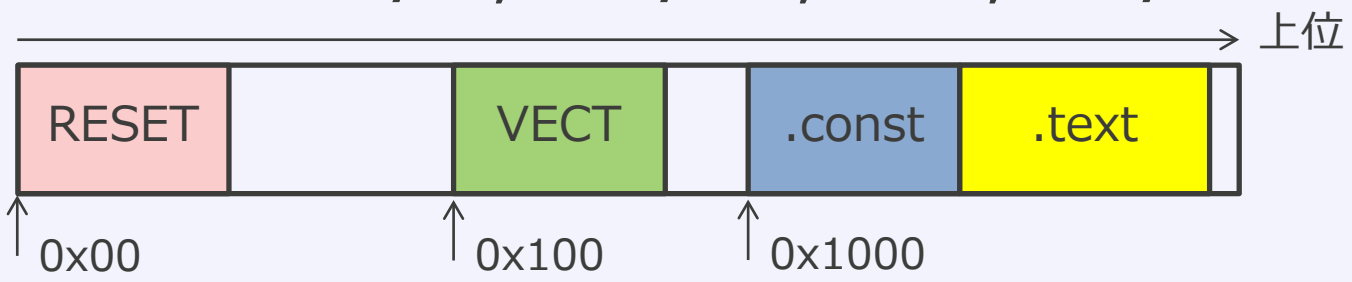
RBASE≠EBASEで使用する場合は、VECTセクションを所望のアドレスに配置させ、
VECTセクションの先頭アドレスをEBASEに設定してください。
VECTセクションを0x100番地に配置したい場合は、右記の例のような定義してください。

セクションの配置アドレスはリンカオプション”-start”で指定します。

```
-start=section1,section2,.../address
```

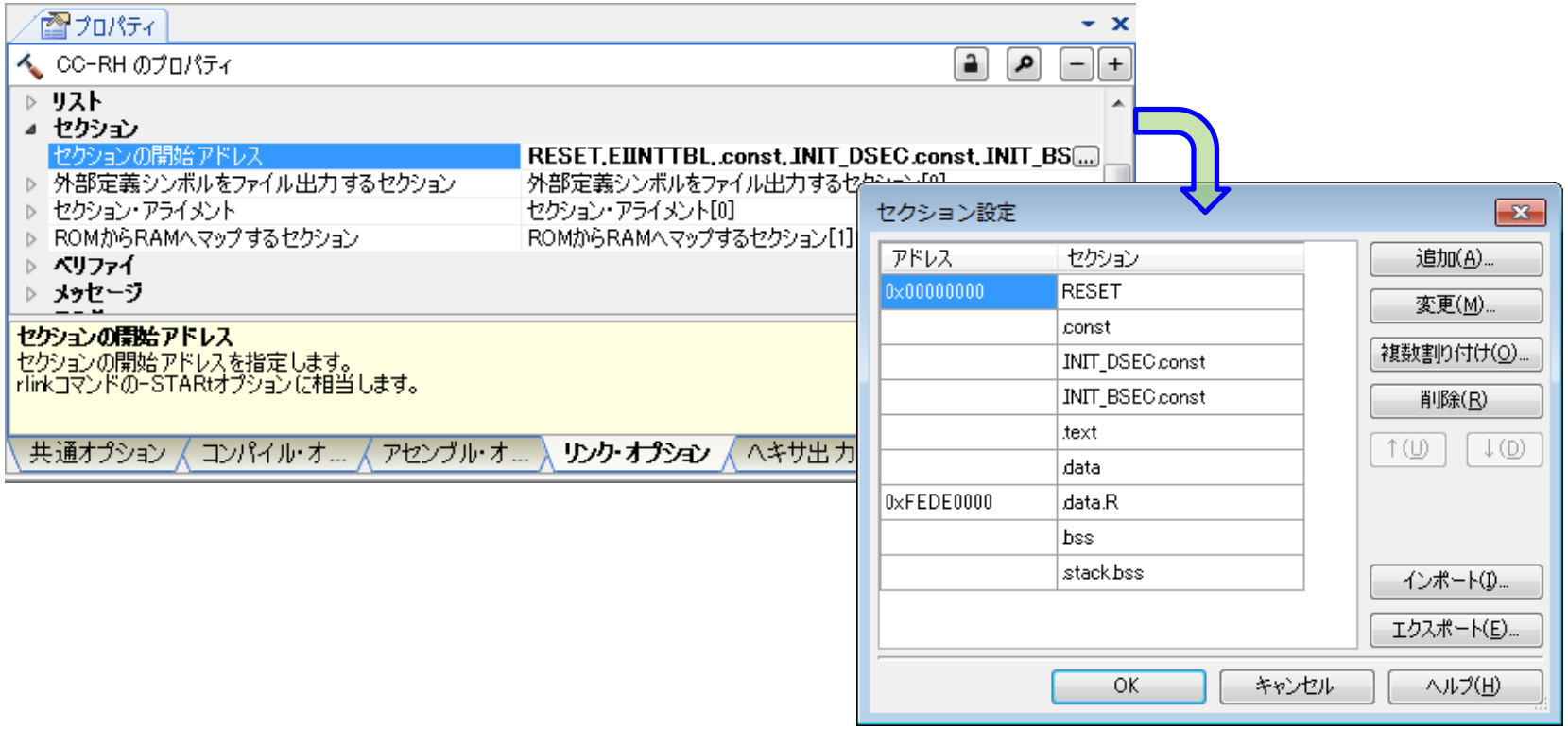
例：0x100番地から上位方向にVECTセクションを、0x1000番地から上位方向に.const、.textセクションを配置させる場合

-start= RESET/00,VECT/100,.const,.text/1000



直接ベクタ方式のベクタ定義5

統合開発環境CS+上からGUI操作により-startオプションを指定することも可能です。
[リンク・オプション]タブ => [セクション]カテゴリ =>[セクションの開始アドレス]から[セクション設定]ダイアログを起ち上げて指定してください。



直接ベクタ方式のベクタ定義6

RBASE≠EBASEで使用する場合は、VECTセクションの先頭アドレスをINTBPに設定する必要があります。セクションの先頭アドレスは特殊シンボルを使用すると便利です。

参考：特殊シンボル

アセンブラにおいて
- セクション名に”#__s”を付けると、そのセクションの先頭アドレス
- セクション名に”#__e”を付けると、そのセクションの終端アドレス

例：アセンブラでEBASEの設定方法

```
_set_ebase:
        mov     #__sVECT, r10
        ldsr    r10, 3, 1           ; set EBASE
```

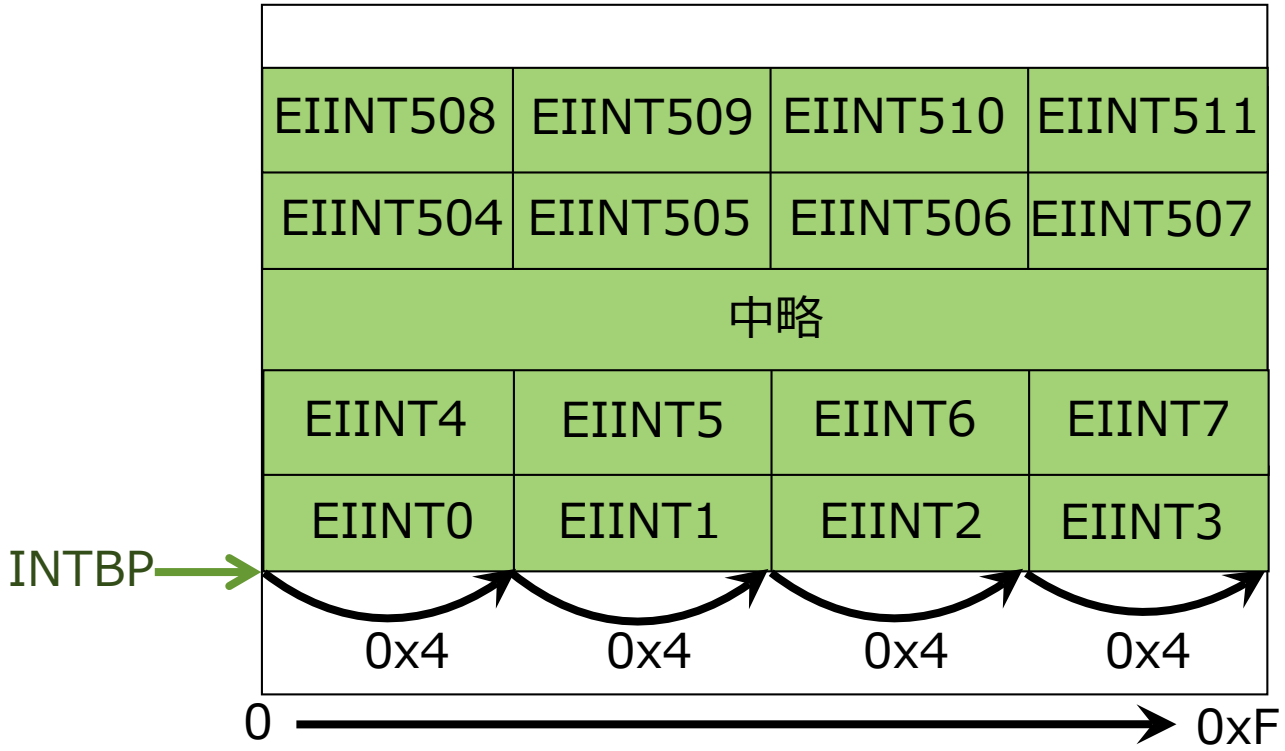
リセット後にVECTセクションの先頭アドレス(#__sVECT)をEBASE(レジスタ番号：SR3, 1)に格納します。

テーブル参照方式のベクタの定義

ハンドラ・アドレスの決定方法

テーブル参照方式のハンドラ・アドレスは
INTBP＋割り込みチャンネル番号*４となります。

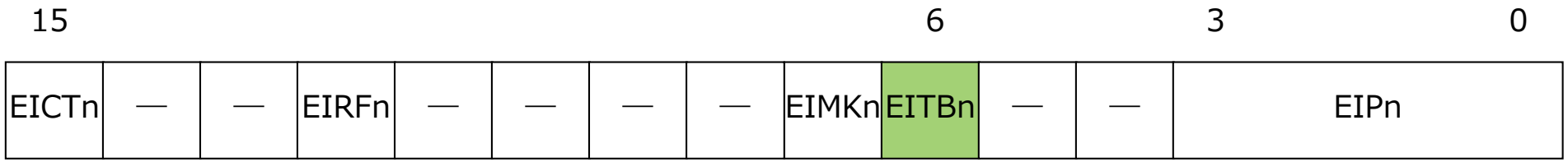
種類	ハンドラ・アドレス
EIINT0	INTBP + 0 * 4
EIINT1	INTBP + 1 * 4
EIINT2	INTBP + 2 * 4
・・・	・・・
EIINT510	INTBP + 510 * 4
EIINT511	INTBP + 511 * 4



テーブル参照方式への変更方法1

EIレベル割り込みEIINTnは、直接分岐方式とテーブル参照方式のいずれかを選択可能です。デフォルトでは直接分岐方式となっています。テーブル参照方式に変更するには割り込み制御レジスタEICnで選択します。

EICn (n：チャネル番号)



【EITBn】　割り込みベクタ方式選択ビット

0 ： 直接分岐方式

1 ： テーブル参照方式

テーブル参照方式への変更方法2

例：EIC0レジスタのアドレスが0xFFFEEA00のマイコンの

EIINT0～EIINT3の割り込みをアセンブラでテーブル参照方式に変更する場合

```
_init_eiint:
        mov     0xFFFEEA00, r10         ; get EIC0 register address
        set1    6, 0[r10]               ; set EIINT0 as table reference
        set1    6, 2[r10]               ; set EIINT1 as table reference
        set1    6, 4[r10]               ; set EIINT2 as table reference
        set1    6, 6[r10]               ; set EIINT3 as table reference
```

EIINT0の割り込み制御レジスタEIC0のアドレスをベースアドレスにして、そのアドレスからのオフセットにより、それぞれの割り込みベクタ選択ビットにアクセスすると便利です。

テーブル参照方式のベクタ定義1

ベクタは各チャネルごとに0x4バイト確保されています。アセンブラで.dw疑似命令を使用してテーブル参照方式のベクタを定義してください。

例：4バイト領域を確保し、関数funcのアドレスで初期化する場合

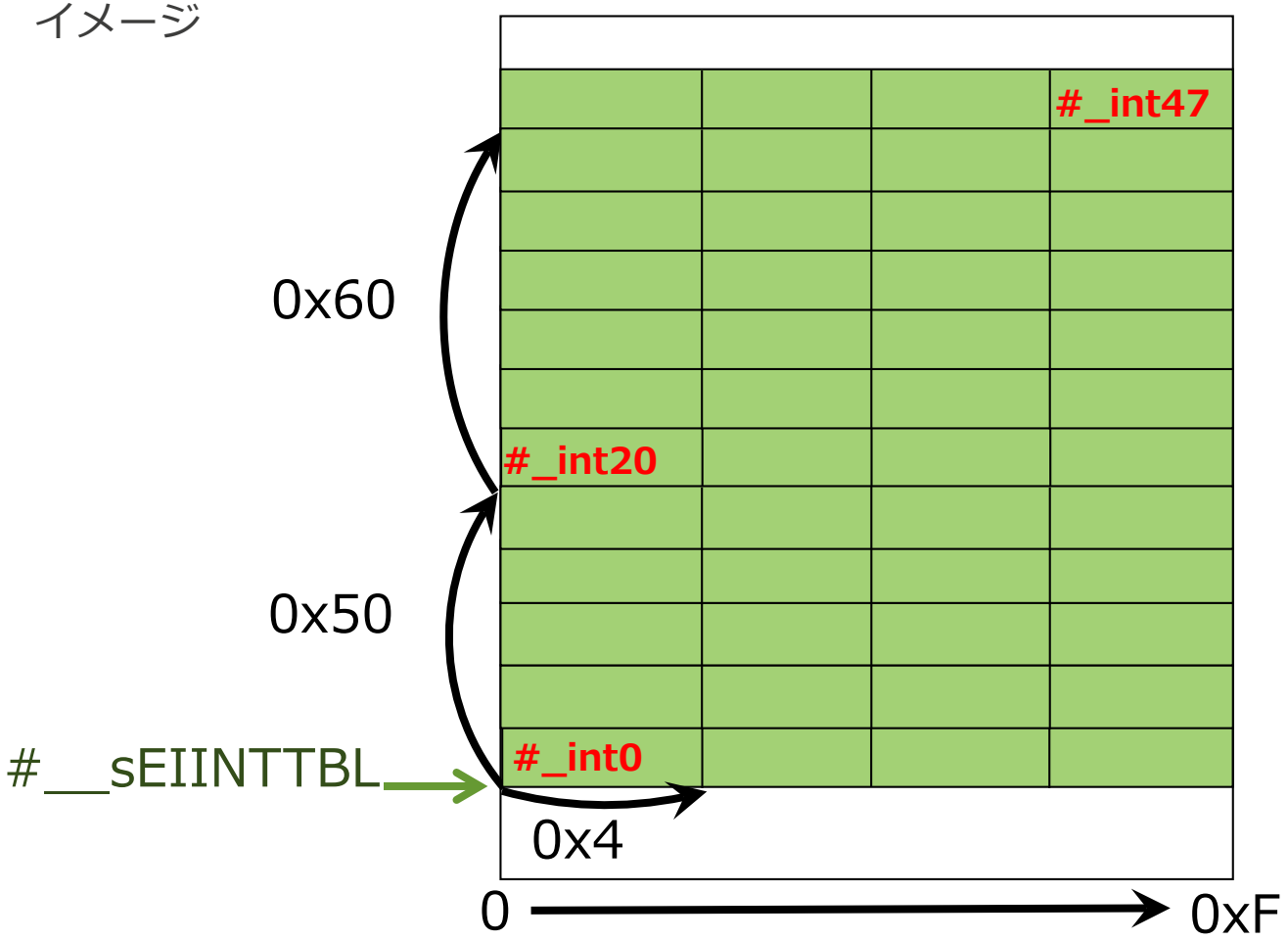
```
        .dw     #_fucn
```

例：EIINT0発生時にint0、EIINT20発生時にint20、EIINT47発生時にint47関数を実行するベクタ定義し、EIINTTBLセクションとする場合

```
        .section "EIINTTBL", const
        .dw     #_int0          ; EIINT0
        .ds     (19 – 0) * 4    ; EIINT1～EIINT19の領域（76byte）を確保
        .dw     #_int20         ; EIINT20
        .ds     (46 - 20) * 4   ; EIINT21～EIINT46の領域（104byte）を確保
        .dw     #_int47         ; EIINT47
```

テーブル参照方式のベクタ定義2

イメージ




```
.section "EIINTTBL", const
.dw     #_int0 ; EIINT0
.ds     (19 – 0) * 4
.dw     #_int20         ; EIINT20
.ds     (46 - 20) * 4
.dw     #_int47         ; EIINT47
```

テーブル参照方式のベクタ定義3

EIINTTBLセクションを所望のアドレスに配置させ、EIINTTBLセクションの先頭アドレスをINTBPに設定してください。セクションの先頭アドレスは特殊シンボルを使用すると便利です。

例：アセンブラでINTBPの設定方法

```
_set_intbp:
        mov     #__sEIINTTBL, r10
        ldsr    r10, 4, 1           ; set INTBP
```

リセット後にEIINTTBLセクションの先頭アドレス(#__sEIINTTBL)をINTBP （レジスタ番号：SR4, 1）に格納します。

その他、割り込み制御

割り込みの禁止/許可

CC-RHでは、以下の方法によりソース上からEIレベル割り込みの許可/禁止を制御することが可能です。

- 組み込み関数”__DI()”, ”__EI()”
- #pragma block_interrupt 指令

組み込み関数による制御

組み込み関数” __DI()”, ” __EI()”によりCソース上から割り込みを禁止・許可したい区間を指定することが可能です。

組み込み関数	動作
__DI()	EIレベル・マスカブル割り込み/例外の受け付けを禁止
__EI()	EIレベル・マスカブル割り込み/例外の受け付けを許可

例：組み込み関数の記述方法

```
void func(void) {
   :
  __DI();
  /* 割り込みを禁止して行いたい処理を記述*/
  __EI();
   :
}
```

pragma block_interrupt 指令による制御

#pragma block_interrupt 指令により関数全体を割り込みを禁止にすることが可能です。

#pragma block_interrup 関数名

関数名で指定した関数に対し、EIレベルマスカブル割り込みを禁止します。関数の最初に”__DI ( )” を、最後に”__EI ( )”を記述することもできますが、この場合、CC-RH が出力する入口コードと出口コードに対して割り込みを禁止／許可することができず、関数全体を完全に割り込み禁止にすることができません。
#pragma block_interrupt 指令を用いることで、入口コード実行の直前に割り込みを禁止し、出口コード実行直後に割り込みを許可します。そのため関数全体を完全に割り込み禁止にすることができます。

例： #pragma block_interrupt 指令の記述方法

```
#pragma block_interrupt func
void func(void) {
   :
   /* 割り込みを禁止して行いたい処理を記述*/
   :
}
```

Renesas.com