【注意事項】
RH850 ファミリ用 C コンパイラパッケージ

R20TS0317JJ0100
Rev.1.00
2018.07.16 号

概要

RH850 ファミリ用 C コンパイラパッケージ CC-RH の使用上の注意事項を連絡します。

1. 初期化子の指定がある配列型、構造体型、または共用体型の static 変数宣言に関する注意事項 (No.19)
2. 予約シンボルを使用した場合のアセンブラの注意事項 (No.20)
3. -Xmulti_level オプション指定時に自動変数の初期化子が配置されるセクションに関する注意事項 (No.21)
4. -store_reg オプションに関する注意事項 (No.22)

注： 注意事項の後ろの番号は、注意事項の識別番号です

1. 初期化子の指定がある配列型、構造体型、または共用体型の static 変数宣言に関する注意事項 (No.19)

1.1 該当製品

CC-RH V1.00.00～V1.07.00

1.2 内容

初期化子の指定がある配列型、構造体型、または共用体型の static 変数宣言が関数内に複数あり、下位階層ブロックにある変数の初期化子に、上位階層ブロックにある変数のアドレスを指定すると内部エラーが発生、または不正なコードを生成する可能性があります。

1.3 発生条件

下記(1)～(3)の条件を全て満たし、かつ(4)または(5)のいずれかの条件を満たす場合に内部エラーが発生します。また、下記(1)～(4)の条件を全て満たし、かつ(6)の条件を満たす場合に、不正なコードを生成する可能性があります。

(1) 関数内に(1-1)～(1-3)の条件を全て満たす変数宣言をする。

(1-1) static 指定をする
(1-2) 配列型、構造体型、または共用体型の変数である
(1-3) 初期化子の指定をする

(2) (1)の変数宣言を記述したブロック内で下位階層ブロックに(1)と同じ条件の変数宣言をする。

(3) (2)の変数の初期化子に(1)の変数のアドレスを指定する。

(4) (1)の変数宣言を関数先頭のブロックに記述する。

(5) V1.04.00 以降のバージョンを使用し、かつ(1)の変数宣言を関数先頭のブロックより下位階層ブロックに記述する。

(6) (1)を満たす変数を、(6-1)～(6-4)のいずれかの記述で変数宣言をする。

(6-1) (4)のブロック内で(1)の変数より前に変数宣言を記述する
(6-2) (2)のブロックより前に記述したブロックに変数宣言を記述する

(6-3) (2)のブロック内で(2)の変数より後に変数宣言を記述する
(6-4) V1.03.00 以前のバージョンを使用し、かつ(2)のブロック内で下位階層ブロックに変数宣言を記述する

1.4 発生例

以下に、不正なコードを生成する例を記します。赤文字が発生条件の該当箇所です。

```
1:  typedef struct A {
2:    short *objId;
3:  } TEST_PARAM;
4:
5:  int sub(TEST_PARAM * aParam);
6:  int var;
7:
8:  int main(void)
9:  {
10:   static short obj[] = { 1 };          // 条件(1)(4)
11:   {
12:     static short dmy[] = { 2 };       // 条件(6-2)
13:     var = dmy[0];
14:   }
15:   {
16:     static TEST_PARAM t ={ obj }; // 条件(2)(3)
17:     sub(&t);
18:   }
19:   return 0;
20: }
```

・10 行目：main 関数の先頭ブロック(9～20 行目)内に初期化子の指定がある配列型の static 変数”obj”の宣言があるため、条件(1)と条件(4)に該当します。
・16 行目：10 行目の変数宣言が存在するブロック(9～20 行目)内で、下位階層ブロック(15～18 行目)に初期化子の指定がある構造体型の static 変数”t”の宣言があるため、条件(2)に該当します。また、初期化子として 10 行目の配列型変数”obj”のアドレスを指定しているため、条件(3)に該当します。
・12 行目：10 行目の変数宣言とは別に、初期化子の指定がある配列型の static 変数”dmy”の宣言があり、条件(2)に該当する 16 行目の変数宣言があるブロックより前に記述したブロック(11～14 行目)内に存在しているため、条件(6-2)に該当します。
この場合、構造体型変数”t”は配列型変数”obj”ではなく配列型変数”dmy”のアドレスで初期化されます。

1.5 回避策

以下のいずれかにより回避可能です。

(1) 発生条件(2)の変数宣言で static 指定を外す。
(2) 発生条件(3)の初期化子を代入式に変更する。

1.6 恒久対策

CC-RH V2.00.00 で改修します。（7 月 20 日公開予定）

2. 予約シンボルを使用した場合のアセンブラの注意事項 (No.20)

2.1 該当製品

CC-RH V1.07.00

2.2 内容

予約シンボル__gp_data または__ep_data を定義した同一セクション内のシンボルに対して分離演算子である HIGH, LOW, HIGHW, LOWW, HIGHW1 のいずれかを使用した場合、その演算結果が不正、あるいはエラーとなる可能性があります。

2.3 発生条件

下記(1)～(5)の条件を全て満たす場合に該当します。

(1) アセンブリ・ソースに(1-1)(1-2)いずれかの予約シンボルを定義する。

(1-1) __gp_data
(1-2) __ep_data

(2) (1)の予約シンボルを.public 疑似命令あるいは.extern 疑似命令によりグローバルシンボルとする。

(3) 同じアセンブリ・ソース内で(1)の予約シンボルと同一セクション内に任意のシンボルを定義する。

(4) (3)の任意のシンボルを(4-1)(4-2)いずれかの記号を用いてオフセット参照する。

(4-1) 同一セクションの予約シンボルが(1-1)の場合は"$"を使用した gp オフセット参照
(4-2) 同一セクションの予約シンボルが(1-2)の場合は"%"を使用した ep オフセット参照

(5) (4)のオフセット参照に対して分離演算子を使用している。

2.4 発生例

以下に、発生例を記します。

```
1:  .section .data, data
2:  .public __gp_data                          ;条件(2)
3:  __gp_data:                                 ;条件(1)
4:      .ds 4
5:  _symbol:                                   ;条件(3)
6:  …
7:      addi    highw($_symbol), r4, r10       ;条件(4)(5)
```

3 行目：予約シンボル__gp_data を定義しているため条件(1)に該当します。
2 行目：__gp_data を.public 疑似命令により外部定義宣言しているため条件(2)に該当します。
5 行目：__gp_data と同じ.data セクションにシンボル_symbol を定義しているため条件(3)に該当します。
7 行目：シンボル_symbol を"$"により gp オフセット参照しているため条件(4)に該当します。また、gp オフセット参照に対して分離演算子の HIGHW を使用しているため条件(5)にも該当します。
このとき 7 行目の「highw($_symbol)」の計算結果が不正な値となります。_symbol のアドレスから__gp_data のアドレスの減算値(=0x00000004)に対して上位 16 ビット(=0x0000)を返すのが正しい仕様ですが、下位 16 ビット(=0x0004)を返します。

2.5 回避策

予約シンボル__gp_data 及び__ep_data を定義しないでください。

2.6 恒久対策

CC-RH V2.00.00 で改修します。（7 月 20 日公開予定）

3. -Xmulti_level オプション指定時に自動変数の初期化子が配置されるセクションに関する注意事項 (No.21)

3.1 該当製品

CC-RH V1.00.00～V1.07.00

3.2 内容

-Xmulti_level=1 オプション指定時に、char 型配列の自動変数の初期化子が仕様とは異なるセクションに配置されます。

3.3 発生条件

下記(1)～(3)の条件を全て満たす場合に該当します。(3)の初期化子が仕様とは異なるセクションに配置されます。

(1) -Xmulti_level=1 オプションを指定している。

(2) (2-1)(2-2)いずれかの自動変数を定義している。

(2-1) (signed/unsigned) char 型の配列
(2-2) (signed/unsigned) char 型の配列型を要素に持つ構造体または共用体

(3)　(2)の自動変数の初期化を文字列リテラルで行う。

3.4　発生例

以下に、発生例を記します。赤文字が発生条件の該当箇所です。

```
1:  void func () {
2:    char a[2] = {'1', '2'};
3:    char b[2] = "12";          // 条件(2)(3)
4:  }
```

・2 行目：配列 a の初期化子'1', '2'は.const セクションに配置されます。
・3 行目：配列 b の初期化子"12"は.const セクションに配置するのが正しい仕様ですが、.const.cmn セクションに配置します。

3.5　回避策

初期化を文字列リテラルではなく文字で指定してください。

3.6　恒久対策

CC-RH V2.00.00 で改修します。（7 月 20 日公開予定）

4. -store_reg オプションに関する注意事項 (No.22)

4.1 該当製品

CC-RH V1.06.00～V1.07.00 （Professional 版のみ）

4.2 内容

構造体のメンバへの間接参照が、制御レジスタへの書き込みとして認識されないケースがあります。

その結果、-store_reg オプション (注) を指定しても以下の機能が動作しません。

- 制御レジスタへの書き込み処理の検出
- 制御レジスタ間の同期化処理の挿入制御

注：-store_reg オプションは Professional 版のみ使用できるオプションです。

4.3 発生条件

【条件ｉ】または【条件ii】のいずれかを満たす場合に発生する可能性があります。

【条件ｉ】

下記(1)～(4)の条件を全て満たす場合に、(4)の代入を制御レジスタへの書き込みとして検出できない可能性があります。

(1) 定数を、volatile 修飾した構造体型 ST へのポインタ型にキャストしている。

(2) (1)の構造体型 ST が共用体型 UT のメンバ A を含む。

(3) (2)の共用体型メンバ A は構造体型 ST の先頭メンバではない。

(4) (2)の共用体型メンバ A に対して、下記(a)～(c)のいずれかの条件を満たす代入がある。

(a) A への共用体代入である

(b) A のメンバ B への代入であり、かつ B の型のサイズが共用体型 UT のメンバのうち最大の型のサイズより小さい

(c) A が構造体型 ST2 のメンバ B を持ち、かつ下記(c-1)～(c-3)のいずれかの条件を満たす代入がある

(c-1) B のビットフィールドメンバ C への代入であり、かつ C の型が (unsigned) char / _Bool 型以外である

(c-2) B のビットフィールドメンバ C への代入であり、かつ C の型が (unsigned) char / _Bool 型であり、かつ C の型が共用体型 UT のメンバのうち最大の型のサイズより小さい

(c-3) B のメンバ C への代入であり、かつ構造体 ST2 が (unsigned) int / long 型のメンバを持たない

【条件ii】

下記(5)～(7)の条件を全て満たす場合に、(7)の代入を制御レジスタへの書き込みとして検出できない場合があります。

(5) 定数を、volatile 修飾した構造体型 ST へのポインタ型にキャストしている。

(6) (5)の構造体型 ST が構造体型 ST2 のメンバ A を含む。

(7) (6)の構造体型メンバ A に対して、下記(a)～(b)のいずれかの条件を満たす代入がある。

(a) A への構造体代入である

(b) A のビットフィールドメンバ B への代入であり、かつ B の型が(unsigned) char / _Bool 型 以外である

4.4 発生例

以下に、発生例を記します。赤文字が発生条件の該当箇所です。

```
1:  struct __tag2191
2:  {
3:      unsigned char   EIP384:4;
4:      unsigned char   :2;
5:      unsigned char   EITB384:1;
6:      unsigned char   EIMK384:1;
7:  };
8:
9:  union __tag4514
10: {
11:     unsigned short UINT16;
12:     unsigned char   UINT8[2];
13:     struct __tag2191 BIT;
14: };
15:
16: struct __tag4697
17: {
18:     unsigned char   dummy1812[1];
19:     union __tag4514 EIC384;                                  // 条件(2)(3)
20: };
21:
22: #define INTC2       (*(volatile struct __tag4697 *)0xFFFFB040)  // 条件(1)
23: #pragma register_group 0xFFFFB040, 0xFFFFFFFF, id="INTC2"
24:
25: void fun() {
26:     INTC2.EIC384.BIT.EIMK384 = 1;                            // 条件(4)
27: }
```

・22 行目：定数 0xFFFFB040 を volatile 修飾した構造体型__tag4697 へのポインタ型にキャストしているため、条件(1)に該当します。

・16～20 行目：構造体型__tag4697 が共用体型__tag4514 のメンバ EIC384 を含むため、条件(2)に該当します。

また、EIC384 は構造体型__tag4697 の先頭メンバではないため、条件(3)に該当します。

・26 行目：共用体型メンバ EIC384 は構造体メンバ BIT を持ち、かつ下記を満たすため条件(4)-(C)-(c-2)に該当します。

- BIT のビットフィールドメンバ EIMK384 への代入がある
- EIMK384 の型が unsigned char 型である
- その型が共用体型__tag4514 のメンバの最大の型 unsigned short より小さい

このため、【条件ｉ】に該当します。

INTC2.EIC384.BIT.EIMK384 は、23 行目の#pragma register_group で指定したアドレス（0xFFFFB040～0xFFFFFFFF）の範囲内にあります。そのため、グループ ID が INTC2 の制御レジスタとして検出するのが正しい仕様ですが、検出することができません。

4.5 回避策

同期化処理の挿入が必要な場合は、手動で挿入してください。

4.6 恒久対策

CC-RH V2.00.00 で改修します。（7 月 20 日公開予定）

以上

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2018.07.16	-	新規発行

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）

■総合お問い合わせ先

https://www.renesas.com/contact/