RENESAS ツールニュース

RENESAS TOOL NEWS 2006年05月22日：RSO-H8C-060522D

H8SX, H8S およびH8ファミリ用 C/C++コンパイラパッケージ ご使用上のお願い

H8SX, H8S およびH8ファミリ用C/C++コンパイラパッケージ V.4～V.6 の使用上の 注意事項16件を連絡します。

1. **該当製品**
   V.4.0 ～ V.6.01 Release 01
   製品型名：
   V.4
   Windows版: PS008CAS4-MWR
   Solaris版: PS008CAS4-SLR
   HP-UX版: PS008CAS4-H7R
   V.5
   Windows版: PS008CAS5-MWR
   V.6
   Windows版: R0C40008XSW06R
   Solaris版: R0C40008XSS06R
   HP-UX版: R0C40008XSH06R

2. **内容**

   2.1 **変数の参照に関する注意事項1(H8C-0028)**
   該当バージョン：
   V.6.00 Release 00～V.6.01 Release 01
   内容：
   変数のアドレスをレジスタに設定するときに、下位2バイトまたは下位1バイト のみを設定して、そのため間違った変数のアドレスを参照する場合があり ます。
   発生条件：
   以下のすべての条件を満たす場合に発生することがあります。
   (1) CPUオプションとしてH8SXN, H8SXM, H8SXA, H8SXXまたはAE5 を使用している。(例：-

cpu=h8sxn)

(2) 最適化オプション -optimize=1を使用している。

(3) 以下の(3-1)または(3-2)のいずれかの条件を満たしている。

(3-1)
(a) MOV.sz #const, ERnが複数または、SUB.sz ERn,ERn命令が複数
ある。
(b) (a)のszは、ロングワードサイズまたはワードサイズである。
(c) (a)の命令間で同じレジスタに対して部分的に(上位1または
2バイト、下位1バイトまたは2バイト)値が同じである。
(3-2)
(a) 2つのMOV命令の間にポストインクリメント@ERn+、ポストデクリ
メント@ERn-、プリインクリメント@+ERn、およびプリデクリメント
@-ERnのいずれかの実効アドレス（EA）を使用した命令がある。
(b) (a)の実効アドレスで使用しているレジスタは両MOV命令の
デスティネーションオペランドで設定されているレジスタと
同じレジスタである。
Cソース例：

```
-------------------------------------------------
----
 struct A {
. . . . . .
    union {
       unsigned char c1;
          struct {
             unsigned char DS:1;
             unsigned char CS:1;
             unsigned char BS:1;
. . . . . .
           }UN2;
    }UN1;
 };
```

```
  #define PTRA      (*(volatile struct
A*)0xFFF200)
  void f(){
     . . . . . .
     PTRA.UN1.UN2.DS = 0x00; //(A)
     . . . . . .
     PTRA.UN1.UN2.BS  = ...;
     . . . . . .
     PTRA.UN1.UN2.DS = 0x01; //(B)
  }
-----------------------------------------------
----
```

生成コード：

```
-----------------------------------------------
----
  _f
  . . . . . .
  MOV.L      #H'00FFF211,ER0 ;発生条件(3-1)
                      ;Cソース例の(A)の生成コードで
ある
                      ;p->UN1.UN2.DSのアドレスを
ロード
  MOV.B      #1:8,R1L
  . . . . . .
  MOV.B      R3L,R0L
  EXTU.L     #2,ER0          ;ER0に別の値を設定
  MOV.L      ER0,ER4
  . . . . . .
  MOV.W       #H'F211:16,R0 ;発生条件(3-1)
                     ;Cソース例の(B) の生成コードで
ある
                     ;アドレス下位2バイト分をロード
  BSET.B     #7,@ER0
-----------------------------------------------
----
```

回避策：

以下のいずれかの方法で回避してください。

(1)　#pragma option nooptimizeを使用してこの問題が発生する関数に対して 最適化を行わない。

(2)　間違ったコードを生成する箇所の直前に別ファイルで

定義した空関数呼び 出しを挿入する。(モジュール間最適化オプション-goptimize使用時のみ有効な回避策)

例:

```
 void f(){
. . . . . .
   PTRA.UN2.DS = 0x00; //(A)
   . . . . . .
   PTRA.UN4.BS  = ...;
   . . . . . .
   dummy();//dummy()は別ファイルで定義する
   PTRA.UN2.DS = 0x01; //(B)
  }
```

(3) 最適化オプション-optimize=0（最適化なし）を使用する。

2.2 サイズが4バイト以下の構造体または共用体を参照した場合の注意事項 (H8C-0029)

該当バージョン：
V.6.01 Release 01
内容：
サイズが4バイト以下の構造体または共用体のメンバを参照すると間違った値 を参照する場合があります。また、前述のメンバを演算の対象として使用 すると、演算しない場合があります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1) CPUオプションとして2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXXまたはAE5を使用している。(例： -cpu=2000n)
ただし出力オブジェクト互換オプション-legacy=v4を使用しているときは、2000N, 2000A, 2600N, および2600Aは該当しません。

(2) サイズが4バイト以下の構造体または共用体を定義および宣言している。

(3) (2)の構造体または共用体に、構造体または共用体のサイズと異なるサイズの構造体または共用体メンバが含まれている。

(4)　(2)の構造体または共用体にはvolatile修飾子が付加されていない。

(5)　(2)の構造体または共用体型変数は、レジスタに割りついているか、または、構造体パラメタのレジスタ割り付けオプション-structregを使用して、ある関数の実引数として使用している。

発生例：

```
-----------------------------------------------------
 void main(){
   union tag_UNION {          //発生条件(2)および(4)
      signed int      m_si ;  //発生条件(3)
      signed char     m_sc ;
      unsigned short  m_us ;
      long            m_sl ;
   } union_data ;
   union_data.m_si = 0;
   union_data.m_us = 0;
   . . . . . .
   union_data.m_si = union_data.m_us;
   printf(": %u ¥n",union_data.m_us);
   union_data.m_us -= 9952;          //(A)
   printf(": %u ¥n",union_data.m_us);//発生条件(5)
 }
-----------------------------------------------------
```

生成コード：

```
-----------------------------------------------------
 . . . . . .
 mov.l   #_printf:32,er3
 jsr     @er3
 ; 発生例の(A)のunion_data.m_us -= 9952;の演算をしない
 mov.l   #C_00000018:32,er0
 mov.l   er0,@sp
 mov.l   @(6:16,sp),er2
 mov.w   e2,@(4:16,sp)
 jsr     @er3
 mov.l   er2,er0
 mov.b   #h'0f:8,r1l
-----------------------------------------------------
```

回避策：

以下のいずれかの方法で回避してください。

(1)　該当の構造体および共用体をvolatileで修飾する。

(2) 該当の構造体および共用体のサイズを4バイトより大きくする。

2.3 変数の参照に関する注意事項2(H8C-0030)

該当バージョン：
V.6.01 Release 00～V.6.01 Release 01
内容：
変数を参照すると、ディスプレースメントサイズを間違って2ビットで 生成します。
発生条件：
以下の条件をすべて満たす場合に発生します。

(1) CPUオプションとして、H8SXN, H8SXM, H8SXA, H8SXXまたはAE5を使用している。

(2) 最適化オプション-optimize=1を使用している。

(3) MOV命令の値の移動先または演算結果格納先となるデスティネーションが"@(シンボル,ERn)"または"@(シンボル+定数,ERn)"である。

発生例：

```
----------------------------------------------------
 typedef struct {
    char *c1;
    char *c2;
 }ST;

 ST st1[3];

 void func(long l1,long l2){
    ST st2[5];
    st1[l1].c2 = st2[l2].c2;
 }
----------------------------------------------------
```

生成コード：

```
----------------------------------------------------
 _func:
   sub.w   #h'0028:16,r7
   shll.l  #3:5,er1
   add.l   sp,er1
   shll.l  #3:5,er0
   mov.l   @(4:16,er1),@(_st1+4:2,er0) ;発生条件(3)
   ;間違って"@(_st1+4:2,er0)"になっている。正しく
```

は"@(_st1+4:32,er0)"。
```
   add.w   #h'0028:16,r7
   rts
```

回避策：
以下のいずれかの方法で回避してください。

(1) #pragma option nooptimizeを使用して該当する関数に対して最適化を行わない。

(2) volatile修飾子のついた変数に代入してから代入および演算を行う。

例:
```
typedef struct {
   char *c1;
   char *c2;
}ST;

ST st1[3];
volatile char *dummy;

void func(long l1,long l2){
   ST st2[5];
   dummy = st2[l2].c2;  //volatile修飾子を持つ変数に代入してから
   st1[l1].c2 = dummy;  // 代入および演算を行う。
}
```

(3) 最適化オプション-optimize=0（最適化なし）を使用する。

2.4 ビットフィールドメンバへの定数代入に関する注意事項 (H8C-0031)

該当バージョン：
V.6.00 Release 00～V.6.01 Release 01
内容：
ビットフィールドメンバに定数を代入するときに、 すでに設定されているビットフィールドのメンバの値を初期化せずに、そのままメンバの値と定数をOR演算する場合があります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1)　以下の(A)または(B)を満たしている。

(A) V.6.00を使用している場合
　CPUオプションにH8SXN, H8SXM, H8SXAまたはH8SXXを使用している。
(B) V.6.01 を使用している場合
　CPUオプションに2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM,
　H8SXA, H8SXXまたはAE5を使用している。
　ただし出力オブジェクト互換オプション-legacy=v4を使用しているときは、
　2000N, 2000A, 2600N, および2600Aは該当しません。

(2)　unsigned longまたはsigned long型のビットフィールドメンバを持つ構造体または共用体が宣言されている。

(3)　(2)のビットフィールドメンバのビットサイズは3から31ビットである。

(4)　(2)のビットフィールドメンバの内の一つは15ビットと16ビットにまたがって配置している。

(5)　(2)のビットフィールドメンバに-((2**(発生条件(3)のビットサイズ))-1)の定数値を代入している。

注: **は累乗を示す。

発生例：

```
----------------------------------------------------
struct st{        //発生条件(2)
   long l1:14;
   long l2:3;    //発生条件(3)および(4)
}st1 = { 0, 2 };

void main(void){
   st1.l2 = -7;  //発生条件(5)
}
----------------------------------------------------
```

生成コード：

```
----------------------------------------------------
_main:
mov.l   @_st1:32,er1
or.l    #h'00008000:32,er1 ;st1.l2を0クリアせずに-7とOR
```

演算している
mov.l　er1,@_st1:32

回避策：
以下の方法で回避してください。

(1)　該当の定数値をvolatile修飾している変数に代入し、
その変数から値を代入する。

例：
```
struct st{
   long l1:14;
   long l2:3;
}st1 = { 0, 2 };

volatile long l;
void main(void){
   l = -7;     // volatile修飾子を持つ変数に代入し、
   st1.l2 = l; // その変数から値を代入する
}
```

2.5　同じ構造体型または共用体型のメンバを複数持つ共用体に関する注意事項 (H8C-0032)

該当バージョン：
V.6.00 Release 00～V.6.01 Release 01
内容：
同じ構造体型または共用体型のメンバを複数持つ共用体型変数を使用した場合、 共用体先頭からの間違った共用体メンバオフセット値でメンバを参照する場合 があります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1)　以下の(A)または(B)を満たしている。

(A) V.6.00を使用している場合
　CPUオプションにH8SXN, H8SXM, H8SXAまたはH8SXXを使用している。
(B) V.6.01を使用している場合
　CPUオプションに2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA,
　H8SXXまたはAE5を使用している。
　ただし出力オブジェクト互換オプション-legacy=v4を使用している

ときは、2000N, 2000A, 2600N および2600Aは該当しません。

(2) 共用体型を使用している。

(3) (2)の共用体は構造体型または共用体型のメンバを持っている。

(4) (2)の共用体は(3)のメンバとは別に(3)のメンバと同じ構造体型または共用体型のメンバを持っている。

(5) 共用体先頭から(3)と(4)のメンバまでのオフセット値が同じである。

(6) (3)と(4)のメンバのうち、宣言順が2番目以降のメンバの参照もしくは定義を行っている。

発生例：

```
-------------------------------------------------
typedef struct {
   unsigned int x;
} ST;
typedef union {         //発生条件(2)および(3)
   struct {
      unsigned char a;
      ST s2[1];
   } st1;              //発生条件(4)
   struct {            //発生条件(5)
      unsigned char c;
      ST s3;
   } st2;              //発生条件(4)
} UN;
void func(){
   volatile int a=0;
   UN u;
   f(u.st2.s3.x);      //発生条件(6)
}
-------------------------------------------------
```

回避策：

以下の方法で回避してください。

(1) メンバ名は同じで別の名前の構造体型または共用体型を定義し、 該当する構造体型または共用体型メンバを別の名前で型宣言して ください。

例:

```
typedef struct {
   unsigned int x;
} ST;
typedef struct {
   unsigned int x;
} ST1;
typedef union {
   struct {
      unsigned char a;
      ST s2[1];
   } st1;
   struct {
      unsigned char c;
      ST1 s3;
   } st2;
} UN;
```

2.6 式と整数定数を乗算した結果を同じ整数定数で除算する場合の注意事項 (H8C-0033)

該当バージョン：
V.6.00 Release 00～V.6.01 Release 01
内容：
式と整数定数を乗算した結果を、同じ整数定数で除算したときに、演算結果が 正しくない場合があります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1) 以下の、(A)または(B)を満たしている。

(A) V.6.00を使用している場合
CPUオプションにH8SXN, H8SXM, H8SXAまたはH8SXXを使用している。
(B) V.6.01を使用している場合
CPUオプションに2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM,
H8SXA, H8SXXまたはAE5を使用している。
ただし出力オブジェクト互換オプション-legacy=v4を使用している
ときは、2000N, 2000A, 2600N,および2600Aは該当しません。

(2) 符号無し整数型の式と0以外の整数定数の乗算式が存在する。

(3)　(2)の乗算結果を、(2)の定数で除算している。

(4)　(2)の乗算結果の値がその乗算式の型の最大値を超える。

発生例：

```
-------------------------------------------------
 unsigned long a=65536ul;
 unsigned long b;
 void func() {
    b=(65537ul*a)/65537ul;  // b=0
((65537*65536)/65537→0/65537=0) が
                        // 正しい結果だが、b=a (=65536)に
                        // 置き換えられる
 }
-------------------------------------------------
```

回避策：

以下の方法で回避してください。

(1)　該当する乗算式を、volatile修飾された変数に代入して、この変数を除算の被除数とする。

例：

```
void func() {
   volatile unsigned long c = 65537ul*a;
   b = c / 65537;
}
```

2.7　volatile修飾した制御変数を持つ繰り返し文に関する注意事項 (H8C-0034)

該当バージョン：

V.6.00 Release 00～V.6.01 Release 01

内容：

volatile修飾した制御変数を持つ繰り返し文の繰り返し回数が正しく ない場合があります。

発生条件：

以下の条件をすべて満たす場合に発生することがあります。

(1)　以下の(A)または(B)を満たしている。

(A) V.6.00を使用している場合
　CPUオプションにH8SXN, H8SXM, H8SXAまたはH8SXXを使用している。

(B) V.6.01 を使用している場合
CPUオプションに2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXXまたはAE5を使用している。
ただし出力オブジェクト互換オプション-legacy=v4を使用している
ときは、2000N, 2000A, 2600N,および2600Aは該当しません。

(2) 最適化オプション-optimize=1を使用している。

(3) 繰り返し文が存在する。

(4) (3)の繰り返し文は整数型の制御変数を持つ。

(5) (4)の制御変数をvolatile修飾している。または(4)の制御変数は外部変数でかつvolatile=1オプションを使用している。

(6) (4)の制御変数の更新式は加算式もしくは減算式である。

(7) (3)の繰り返し文の中に関数呼び出しがある。
もしくは(4)の制御変数をさすポインタ型変数が存在し、かつ(3)の繰り返し文の中でこのポインタ変数をリセットしている。

発生例：

```
-----------------------------------------------------
 extern void sub();
 volatile int i;  //発生条件(5)
 void func() {
    i = 1;
    while (i) {  //発生条件(3)および(4)
       i--;     //発生条件(6)
       sub();   //発生条件(7)
              // 関数呼び出し先でiがリセットされる可能性があるが、
              // 繰り返し回数は常に1回になる
    }
    return;
 }
-----------------------------------------------------
```

回避策：
以下のいずれかの方法で回避してください。

(1) 発生条件(6)の更新式の後に発生条件(4)の制御変数を参照する式を追加する。

(2) #pragma option nooptimizeを使用して該当する関数に対して最適化を行わない。

(3) 最適化オプション-optimize=0 （最適化なし）を使用する。

2.8 繰り返し文の制御式中にカンマ演算子を使用した場合の注意事項(H8C-0035)

該当バージョン：
V.6.00 Release 00～V.6.01 Release 01
内容：
繰り返し文の制御式中にカンマ演算子を使用した場合に、式の評価を右側から行う場合があります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1) 以下の、(A)または(B)を満たしている。

(A) V.6.00を使用している場合
CPUオプションにH8SXN, H8SXM, H8SXAまたはH8SXXを使用している。
(B) V.6.01を使用している場合
CPUオプションに2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM,
H8SXA, H8SXXまたはAE5を使用している。
ただし出力オブジェクト互換オプション-legacy=v4を使用している
ときは、2000N, 2000A, 2600Nおよび2600Aは該当しません。

(2) 最適化オプション-optimize=1を使用している。

(3) 繰り返し文が存在する。

(4) (3)の繰り返し文内に整数型の制御変数が存在する。

(5) (4)の制御変数の更新式は加算式もしくは減算式である。

(6) (3)の繰り返し文の繰り返し回数は1回である。

(7) (3)の繰り返し文の制御式中にカンマで区切られた複数の式が存在する。

発生例：

```
-------------------------------------------------
int A, B;
void func() {
  int i;
  for (i=0; A++, B+=A, i<1; i++) { //発生条件
(3),(4),(5),(6)および(7)
                              // B+=A, A++の順に実行される
    B++;
  }
}
-------------------------------------------------
```

回避策：
以下のいずれかの方法で回避してください。

(1) 繰り返し文を使用しない。

(2) #pragma option nooptimizeを使用して該当する関数に対して最適化を行わない。

(3) 最適化オプション-optimize=0（最適化なし）を使用する。

2.9 volatile修飾子を持つ構造体メンバへの代入に関する注意事項 (H8C-0036)

該当バージョン：
V.4 ～ V.6.01 Release 01
内容：
構造体メンバへの代入で、コンパイラが出力する最適化リンケージエディタ用 付加情報に間違った情報を出力することがあります。そのため最適化リン ケージエディタのレジスタ退避・回復最適化を有効にしたときに間違った代入 コードを生成する場合があります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1) 以下の、(A)または(B)を満たしている。

(A) V.4 ～ V.6.00 Release 03を使用している場合
CPUオプションとして、300, 300HN, 300HA, 2000N, 2000A, 2600N
または2600Aを使用している。
(B) V.6.01を使用している場合
CPUオプションとして、300, 300HN, 300HA,

2000N, 2000A, 2600N
　または2600Aを使用している。
　ただし、2000N, 2000A, 2600Nまたは2600Aの場合は、出力
　オブジェクト互換オプション-legacy=v4を同時に使用している。

(2)　モジュール間最適化用付加情報出力オプション-goptimizeを使用している。

(3)　構造体、共用体、クラスメンバの境界調整数オプション-pack=1を使用している。

(4)　代入式があり、右辺がカンマ式または一次式になっている。

(5)　(4)の代入式は、volatile修飾子のunsigned char およびsigned char型以外のメンバを参照している。

(6)　(4)の代入式の左辺に、volatile修飾子されたメンバがある。

(7)　最適化リンケージエディタで、レジスタ退避・回復の最適化-optimize=registerが有効になっている。

発生例：

```
-----------------------------------------------------
  long a;
  main(){
     struct {
        volatile long          vf1,vf2,vf3;
     } *st;
     . . . . . .
     st->vf1 = ((st->vf2 + st->vf3),(st->vf2 + st-
>vf3));
     //発生条件(3),(4)および(5)
  }
-----------------------------------------------------
```

回避策：

以下のいずれかの方法で回避してください。

(1)　モジュール間最適化付加情報出力オプション-goptimizeを該当ファイルに使用しない。

(2)　最適化リンケージエディタで、レジスタ退避・回復の最適化-optimize=registerを無効にする。

2.10 ポインタサイズが2バイト時の添え字が符号無し整数型変数である配列のオーバーフローに関する注意事項(H8C-0037)

該当バージョン：
V.6.00 Release 00～V.6.01 Release 01
内容：
ポインタサイズ指定オプション-ptr16を使用時またはポインタサイズ 2バイト指定キーワード__ptr16を使用したとき、unsigned shortまたは unsigned int型の添え字のオーバーフローが発生した場合、間違った領域を 参照することがあります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1) 以下の(A)または(B)を満たしている。

(A) V.6.00を使用している場合
CPUオプションにH8SXN, H8SXM, H8SXAまたはH8SXX
を使用している。
(B) V.6.01を使用している場合
CPUオプションに2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM,
H8SXA, H8SXXまたはAE5を使用している。
ただし出力オブジェクト互換オプション-legacy=v4を使用している
ときは、2000N, 2000A, 2600Nおよび2600Aは該当しません。

(2) ポインタサイズ指定オプション-ptr16を使用している。または、ポインタサイズ2バイト指定キーワード__ptr16を使用している。

(3) 配列の添え字が定数を含むunsigned short、またはunsigned int型の式で、その式がオーバーフローする。

発生例：

```
---------------------------------------------------
#include <stdio.h>
unsigned int i=15;
char xcary[15]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14};
void main(void){
  unsigned short soeji;
```

```
    soeji = (unsigned short)(i+0xFFFF); //発生条件(3)
    if(xcary[soeji] == 14)  printf("Hello¥n");
    //または
    if(*(xcary+soeji) == 14)  printf("Hello¥n");
}
```

回避策：
以下のいずれかの方法で回避してください。

(1) ポインタサイズ指定オプション-ptr16を使用しない。
かつ、ポインタサイズ2バイト指定キーワード
__ptr16キーワードを使用しない。

(2) 添え字へ代入する変数にオーバーフローする式を設定しない

例

```
#include <stdio.h>

unsigned int i;
char xcary[15];
void main(void){
    unsigned short soeji;

    soeji = (unsigned short)(i-1);  // オーバーフローしない式に
                                    変更する
    if(xcary[soeji] == 14)  printf("Hello¥n");
}
```

2.11 繰り返し文のループ展開の定数参照に関する注意事項(H8C-0038)

該当バージョン：
V.6.01 Release 00～V.6.01 Release 01
内容：
繰り返し文のループ展開を行った場合に、間違った定数を参照する場合 があります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1) CPUオプションとして、300, 300HN, 2000Nまたは2600Nを使用している。
ただし2000Nまたは2600Nの場合は、出力オブジェ

クト互換オプション-legacy=v4を同時に使用している。

(2) 最適化オプション-optimize=1を使用している。

(3) スピード優先最適化オプション-speed=loop=[1|2]が有効になっている。

(4) 繰り返し文内で"定数-変数"の式がある。

(5) (4)の変数は、(4)の繰り返し文内で加算される。

発生例：

```
----------------------------------------------------
void p027(){
  long a;
  long b=11;

  a=2147483640;
  while(a<2147483647){
     b+=sub27(2147483647-a);  //発生条件(4)
     ++a;                     //発生条件(5)
  }
}
----------------------------------------------------
```

生成コード

```
----------------------------------------------------
jsr    @_sub27:16
inc.l  #1,er5
add.l  #h'0000ffff:32,er4 ;正しくは、イミディエイト値がh'ffffffff。
cmp.l  #h'7fffffff:32,er5
----------------------------------------------------
```

回避策：

以下のいずれかの方法で回避してください。

(1) 繰り返し文の制御変数にvolatile修飾する。

例

```
void func(){
  volatile long a;
  long b=11;
```

(2) #pragma option nooptimizeを使用して該当する関数に対して最適化を 行わない。

(3)　最適化オプション-optimize=0（最適化なし）を使用する。

2.12　間違った構造体配列または共用体配列要素へのアクセス (H8C-0039)

該当バージョン：
V.6.00 Release 00～V.6.01 Release 01
内容：
extern宣言された構造体型または共用体型の配列の宣言より後にその構造体 または共用体の定義がある場合に、その配列の先頭以外の要素をアクセス すると、配列の先頭要素をアクセスするコードを生成します。
発生条件：
以下の条件をすべて満たす場合に発生します。

(1)　以下の(A)または(B)を満たしている。

(A) V.6.00を使用している場合
　CPUオプションにH8SXN, H8SXM, H8SXA, H8SXXまたはAE5
　を使用している。
(B) V.6.01を使用している場合
　CPUオプションに2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM,
　H8SXA, H8SXXまたはAE5を使用している。

ただし出力オブジェクト互換オプション-legacy=v4を使用しているときは、2000N, 2000A, 2600Nおよび2600Aは該当しません。

(2)　不完全構造体型の配列を宣言している。

(3)　(2)の構造体定義をその後に記述している。

(4)　(2)の配列要素のアクセスを記述している。

発生例：

```
----------------------------------------------------
extern struct TBL g[3]; //発生条件(2)
struct TBL {            //発生条件(3)
  int m;
};
struct TBL tbl;
void func()
{
```

```
    tbl.m = g[1].m;     //発生条件(4)
    // g[0].mを間違ってアクセスする
}
```

回避策：
以下の方法で回避してください。

(1) 構造体定義を先に記述する。

例

```
struct S {
  int m;
};
extern struct S a[10];
```

2.13 評価結果が真または偽になる式をリターン値にした場合の注意事項(H8C-0040)

該当バージョン：
V.4 ～ V.6.01 Release 01
内容：
条件式の結果をリターン値にしたときに、リターン型に合わせた型変換を せずにリターン値を返却します。
発生条件：
以下の条件をすべて満たす場合に発生します。

(1) 以下の(A)または(B)を満たしている。

(A) V.4 ～ V.6.00 Release 03を使用している場合
CPUオプションとして、300, 300HN, 300HA, 2000N, 2000A, 2600N
または2600Aを使用している。
(B) V.6.01 を使用している場合
CPUオプションとして、300, 300HN, 300HA, 2000N, 2000A, 2600N
または2600Aを使用している。
ただし、2000N, 2000A, 2600Nまたは2600Aの場合は、出力オブ
ジェクト互換オプション-legacy=v4を使用している。

(2) 最適化オプション -optimize=0（最適化なし）を使用している。

(3)　条件式の結果を返却値としている。

(4)　(3)のリターン値の型は、long型、float型または
double型を使用している。
CPUオプションが300HA, 2000A, 2600Aの場合は、
上記に加えポインタ型を使用している

発生例：

```
-----------------------------------------------
----
 int wff1,wff2;

 float func(){          // 発生条件(4)
   return(wff1 == wff2); // 発生条件(3)
 }
-----------------------------------------------
----
```

回避策：
以下の方法で回避してください。

(1)　条件式の結果をローカル変数に代入し、そのローカル
変数値を返却 する。

例：

```
int wff1,wff2;

float func(){
  int rtn; //条件式の結果を代入するローカル変数を
定義する。

  rtn = (wff1 == wff2);
  return(rtn); //ローカル変数の値を返却値にする。
}
```

2.14　下位ビットから割り付けられている共用体ビットフィールドメンバ 参照に関する注意事項(H8C-0041)

該当バージョン：
V.6.01 Release 00～V.6.01 Release 01
内容：
下位ビットから割り付けられている共用体ビットフィールドメンバの 上位に8ビット以上の空き領域がある場合に、間違った領域を参照します。
発生条件：

以下の条件をすべて満たす場合に発生します。

(1) CPUオプションに2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXXまたはAE5を使用している。
ただし出力オブジェクト互換オプション-legacy=v4を使用しているときは、2000N, 2000A, 2600Nおよび2600Aは該当しません。

(2) ビットフィールド並び順オプション-bit_order=right
または
#pragma bit_order rightを使用している。

(3) 共用体を宣言している。

(4) (3)の共用体メンバにビットフィールドメンバがある。

(5) (3)の共用体型変数は、外部変数、または、static指定した内部変数で宣言されている。

(6) (4)のビットフィールドメンバは以下のいずれかを満たしている

(6-1) 構造体、共用体、クラスメンバの境界調整数オプション-pack=1が
使用されている、または、#pragma pack 1が(3)の共用体に対して
使用されている場合
-signed/unsigned short型 ビットサイズ: 1から8ビット
-signed/unsigned int型　ビットサイズ: 1から8ビット
-signed/unsigned long型　ビットサイズ: 1から24ビット
(6-2) 構造体、共用体、クラスメンバの境界調整数オプション-pack=1が
使用されていない。かつ、#pragma pack 1が(3)の共用体に対して
使用されていない場合
-signed/unsigned long型　ビットサイズ: 1から16ビット

発生例：

```
--------------------------------------------------
#include <stdio.h>
typedef union {        // 発生条件(3)
```

```
   unsigned short us:4;// 発生条件(4),(6)
 } UNI1;              // 発生条件(5)

UNI1 uni1 = { 8 };

void main(){
   if(uni1.us == 8)
      printf("OK¥n");
}
```

生成コード：

```
SUBS    #4,sp
MOV.B   @_uni1+2:32,r0l ;"@_uni1+2"を間違って参照し
ている。
                    正しくは"@_uni1+1"。
AND.B   #h'0f:8,r0l
```

回避策：

以下の方法で回避してください。

(1) ビットフィールドメンバをstatic指定のない同じ共用体型の局所変数に代入してアクセスする。

例：

```
#include <stdio.h>
typedef union {
   unsigned short us:4;
} UNI1;

UNI1 uni1 = { 8 };

void main(){
   UNI local_uni1=uni1;
   if(local_uni1.us == 8)
      printf("OK¥n");
}
```

2.15 除数が定数の最小値である除算に関する注意事項(H8C-0042)

該当バージョン：

V.4 ～ V.6.01 Release 01

内容：
除算の除数および被除数が最小値同士の場合、間違った除算結果になります。
発生条件：
以下の条件をすべて満たす場合に発生します。

(1) 以下の、(A)または(B)を満たしている。

(A) V.4 ～ V.6.00 Release 03を使用している場合
CPUオプションとして、300, 300HN, 300HA, 2000N, 2000A, 2600N
または2600Aを使用している。
(B) V.6.01を使用している場合
CPUオプションとして、300, 300HN, 300HA, 2000N, 2000A, 2600N
または2600Aを使用している。ただし、2000N, 2000A, 2600Nまたは
2600Aの場合は、出力オブジェクト互換オプション-legacy=v4を使用
している。

(2) 最適化オプション-optimize=0を使用している。

(3) スピード優先最適化オプション(-speedまたは-speed=expression)を使用している。

(4) 除数の型がsigned short, signed intまたは、signed long型である除算が存在する。

(5) (4)の除数は定数でかつその値は定数の型で表現できる最小の値である。

(6) 被除数には除数と同値が設定されている。

発生例：

```
-------------------------------------------------
short rtn;

void func(){
  short s;

  s = 0x8000;           //発生条件(6)
  rtn = s/(short)0x8000; //発生条件(4),(5)および(6)
  //間違ってrtnが-1になる。正しくはrtnは1。
}
-------------------------------------------------
```

回避策：
以下のいずれかの方法で回避してください。

(1) 最適化オプション-optimize=1を使用する。

(2) スピード優先最適化(-speedまたは-speed=expression)を使用しない。

(3) 除数の定数を変数に代入して演算する。

発生例の場合の回避例：

```
short rtn;

void func(){
  short s;
  short tmp;

  s = 0x8000;
  tmp = (short)0x8000;  //除数を変数に代入する。
  rtn = s / tmp;        //変数同士で演算する。
}
```

2.16 更新された変数の値の参照に関する注意事項(H8C-0043)

該当バージョン：
V.4 ～ V.6.01 Release 01
内容：
メモリに割りついている変数の値が書き換わったとき、メモリからロード せずに正しくない値を参照することがあります。
発生条件：
以下の条件をすべて満たす場合に発生することがあります。

(1) 最適化オプション-optimize=1を使用している。

(2) 以下の(a)または(b)のいずれかを満たす。

(a) ブロック転送命令オプション-eepmovを使用し、eepmov命令が出力
  されている。
(b) ブロック転送命令を出力する組み込み関数
eepmov, eepmovb,
  eepmovw, eepmovi, eepromb, eepromw,
eepromb_exr, eepromw_exr,
  movmdb, movmdw, movmdlおよびmovsdのい

ずれかが使用されている。

発生例：

```
-----------------------------------------------------
#include <stdio.h>
struct ST {
   char c[2];
};
struct ST  st1,st2,st3;
void sub1()
{
   struct ST  *pst;

   pst = &st3;
   st2 = *pst;
   *pst = st1;
}
void main()
{
   st1.c[0] = 1;
   st2.c[0] = 2;
   st3.c[0] = 3;
   sub1();
   if ( st1.c[0]==1 && st2.c[0]==3 && st3.c[0]==1)
printf("OK¥n");
   . . . . . .
}
-----------------------------------------------------
```

生成コード：

```
-----------------------------------------------------
MOV.B       #3,R0H
MOV.B       R0H,@_st3:32
MOV.L       #_st3,ER6
MOV.L       ER4,ER1
MOV.L       ER6,ER2
. . . . . .
EEPMOV.B ;発生条件(2)eepmov命令
. . . . . .
if ( st1.c[0]==1 && st2.c[0]==3 && st3.c[0]==1)
printf("** OK **¥n");
MOV.B       @ER5,R5L
CMP.B       R0L,R5L
BNE         L55:8
```

```
MOV.B       @ER4,R4L
CMP.B       R0H,R4L
BNE         L55:8
MOV.B       R0H,R1L ; メモリから読み込まず、間違ってレジスタから
                    ; コピーしている。
CMP.B       R0L,R1L
```

回避策：
以下のいずれかの方法で回避してください。

(1) #pragma option nooptimizeを使用して該当する箇所の関数に対して最適化を行わない。

(2) ブロック転送命令オプション-eepmovを使用している場合は、本オプションを使用しない。

(3) ブロック転送命令を出力する組み込み関数を使用している場合は、その組み込み関数を使用しない。

(4) 間違ったコードを生成する箇所の直前に別ファイルで定義した空関数呼び出しを記述する。ただし、その関数がインライン展開されないようにする。

例：

```
void main(){
   st1.c[0] = 1;
   st2.c[0] = 2;
   st3.c[0] = 3;
   sub1();
   dummy();  //dummy()の定義は別ファイルにする
   if ( st1.c[0]==1 && st2.c[0]==3 && st3.c[0]==1) printf("OK¥n");
   else printf("NG¥n");
}
```

(5) 最適化オプション -optimize=0（最適化なし）を使用する。

3. **恒久対策**
本内容はV.6.01 Release 02 で改修する予定です。

[免責事項]
過去のニュース内容は発行当時の情報をもとにしており、現時点では変更された情報や無効な情報が含まれている場合があります。
ニュース本文中のURLを予告なしに変更または中止することがありますので、あらかじめご承知ください。