RENESAS ツールニュース

RENESAS TOOL NEWS 2008年06月23日：080623/tn1

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 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12件を連絡します。

1. 構造体、共用体、またはそのメンバ変数がキーワード __evenaccessに基づいてアクセスされるように宣言する際の注意事項(H8C-077)
2. ライブラリ関数 scanf()、 fscanf()、および sscanf()を使用する際の注意事項(H8C-078)
3. 式の中で同一変数を複数回使用する場合の注意事項(H8C-079)
4. 境界調整数に1を選択した構造体または共用体のメンバへアクセスする際の注意事項(H8C-080)
5. 空ループ削除オプション del_vacant_loop=1 使用時に多重ループをコーディングする際の注意事項(H8C-081)
6. 同一処理を複数ブロックに記述する際の注意事項(H8C-082)
7. 関数の戻り値を構造体、共用体、またはクラスのメンバ変数に代入する際の注意事項(H8C-083)
8. jmp命令の分岐先のアドレスに関する注意事項(H8A-0001)
9. 絶対セクション内のラベルの記載に関する注意事項(H8A-0002)
10. 引数格納レジスタ数を3で宣言したアセンブラルーチンを呼び出す際の注意事項(LNK-0001)
11. 短絶対アドレッシングモード活用最適化オプション使用時の変数初期値に関する注意事項(LNK-0002)
12. 共通コード統合最適化オプション使用に関する注意事項(LNK-0003)

---

1. 該当製品

H8SX、H8S、およびH8ファミリ用 C/C++コンパイラパッケージ
V.4.0 ～ V.6.02 Release 00
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(C/C++コンパイラ)

2.1 構造体、共用体、またはそのメンバ変数がキーワード __evenaccessに基づいてアクセスされるように宣言する際の注意事項(H8C-077)

該当バージョン:
V.6.01 Release 00 ～ V.6.02 Release 00

現象:
構造体、共用体、またはそのメンバ変数の__evenaccess宣言が無効になることがあります。

発生条件:
以下の条件をすべて満たした場合に発生することがあります。

(1) CPUオプションに 2000N、2000A、2600N、2600A、H8SXN、H8SXM、H8SXA、H8SXX,
H8SXX、AE5、またはRS4を使用している。
(例: コマンドラインでは -cpu=2000N)

(2) 上記オプションで 2000N、2000A、2600N、または2600Aを使用し、出力オブジェクト互換オプション -legacy=v4 を使用していない。

(3) 以下の(A)、(B)、(C)、または(D)の条件を満たしている。

(A) 以下の(A-1)～(A-5)の条件をすべて満たしている。
(A-1) 構造体または共用体を宣言し、そのメンバに構造体または共用体をネストして宣言している。
(A-2) (A-1)のネストしている構造体または共用体のメンバが4バイト以下の整数型である。
(A-3) (A-1)でネストされている構造体または共用体を配列またはポインタ型で宣言している。
(A-4) (A-1)で宣言した構造体または共用体型のポインタを宣言し、__evenaccessを宣言している。あるいは、構造体、共用体、またはそのメンバに__evenaccessを宣言している。
(A-5) (A-1)で宣言した構造体または共用体の領域を確保していない。

(B) 以下の(B-1)～(B-4)の条件をすべて満たしている。
(B-1) 構造体または共用体を宣言し、その境界調整数に1(-pack=1

または#pragma pack 1)を使用している。
(B-2) (B-1)の構造体または共用体のメンバは、構造体型および
ビットフィールドを除く、サイズが2バイトか4バイトのメンバ
である。
(B-3) (B-1)で宣言した構造体または共用体型のポインタを宣言し、
__evenaccessを宣言している。あるいは、構造体、共用体、
またはそのメンバに__evenaccessを宣言している。
(B-4) (B-1)で宣言した構造体または共用体の領域を確保していない。

(C) 以下の(C-1)～(C-3)の条件をすべて満たしている。
(C-1) サイズが4バイト以下の構造体または共用体型の局所変数または
引数を定義している。
(C-2) (C-1)の構造体または共用体がスタック領域に割り当てられてい
る。
(C-3) (C-1)の構造体または共用体のメンバに、__evenaccessを
宣言している。

(D) 以下の(D-1)～(D-3) の条件をすべて満たしている。
(D-1) ビットサイズが1のビットフィールドメンバを持つ構造体または
共用体を宣言している。
(D-2) (D-1)の宣言した構造体または共用体型のポインタに
__evenaccessを宣言している。あるいは、構造体、共用体、
またはそのメンバに__evenaccessを宣言している。
(D-3) 以下の(D-3-1)または(D-3-2)の条件を満たしている。
(D-3-1) ビットフィールドメンバの型がunsigned longである(D-1)
の構造体または共用体型変数を大域変数または局所変数と
して定義しているか、同構造体または共用体型のポインタ
を宣言している。
(D-3-2) ビットフィールドメンバの型がunsigned short、signed
short、unsigned int、またはsigned intである(D-1)の
構造体または共用体のサイズが4バイト以下である。

(4) (3)の構造体または共用体のメンバを参照している。

発生例1:

```
-----------------------------------------------------------------
 struct S {
    struct SS{              //発生条件(3)(A-1)
       unsigned short c:1;   //発生条件(3)(A-2)
    }a[2];                  //発生条件(3)(A-3)
 };
 #define STR (*(struct S __evenaccess *)0x011000)   //発生条件(3)(A-4)
```

```
void func()
{
   STR.a[0].c = 1;   //発生条件(4)
}
```

出力コード:

```
---------------------------------------------------------------------
_func    MOV.L     #H'00011000:32,ER0
         BSET.B    #7:3,@ER0 ;宣言サイズ(2バイト)でアクセスしていない
         RTS
```

発生例2:

```
---------------------------------------------------------------------
#pragma pack 1          //発生条件(3)(B-1)
struct B {
   unsigned char UC1;
   unsigned short US1;  //発生条件(3)(B-2)
};
#define OBJB (*(volatile struct B __evenaccess *)0xFFFF23) //発生条件
                                          //(3)(B-3)
void main(){
   OBJB.US1 = 1; //発生条件(4)
}
```

出力コード:

```
---------------------------------------------------------------------
_main    MOV.W     #H'0001,R1
         MOV.B     R1H,@H'00FFFF24:8  ;宣言サイズ(2バイト)で
         MOV.B     R1L,@H'00FFFF25:8  ;アクセスしていない
```

発生例3:

```
---------------------------------------------------------------------
struct _str {
   __evenaccess long a ;  //発生条件(3)(C-3)
} ;
void func()
{
   volatile struct _str lstr ;  //発生条件(3)(C-1),(3)(C-2)
   if ( lstr.a & 0x80000000 ) { //発生条件(4)
      sub();
   }
}
```

出力コード:

```
-----------------------------------------------------------------
 _func    SUBS.L    #4,ER7
        MOV.B     @ER7,R0L ;宣言したサイズ(4バイト)で
                    ;アクセスしていない
```

発生例4:

```
-----------------------------------------------------------------
//-cpu=h8sxa
struct ST{                          //発生条件(3)(D-1)
 __evenaccess unsigned long DATA:1; //発生条件(3)(D-2)
}st;
int DATA_int;                       //発生条件(3)(D-3-1)
void func (void){
 DATA_int = st.DATA;
}
```

出力コード:

```
-----------------------------------------------------------------
 _func  SUB.L ER1,ER1
       BLD   #7,@st:32 ;宣言したサイズ(4バイト)でアクセスしていない
       BST   #0,R1L
       MOV.W R1,@_DATA_int:32
```

回避策：

(a) 発生条件(1)、(2)、(3)(A)、および(4)を満たしている場合

以下のいずれかの方法で回避してください。

(a-1) 絶対アドレス表記を使用しないで領域を確保した構造体に対して
__evenaccessを宣言する。

回避例:

```
-----------------------------------------------------------------
 struct S {
    struct SS{
       unsigned short c:1;
    }a[2];
 };
 #pragma address STR=0x011000 // #pragma addressで割り付け先を記載
 __evenaccess struct S STR;   // 領域を確保し、__evenaccessを宣言する
 void func()
 {
    STR.a[0].c = 1;
 }
```

(a-2) 参照する構造体メンバに対して__evenaccessを宣言する。

回避例:

```
-----------------------------------------------------------------
 struct S {
    struct SS{
       __evenaccess unsigned short c:1; //参照する構造体の
                                        //メンバに__evenaccess
                                        //を宣言する
    }a[2];
 };

 #define STR (*(struct S *)0x011000) //__evenaccess宣言を外す
-----------------------------------------------------------------
```

(b) 発生条件(1)、(2)、(3)(B)、および(4)を満たしている場合

以下の方法で回避してください。

- サイズが2バイトまたは4バイトの構造体メンバをfloat型を除く
  ビットフィールド形式で宣言して使用する。

回避例:

```
-----------------------------------------------------------------
 struct B {
    unsigned char UC1;
    unsigned short US1:16; //ビットフィールド形式で宣言する
 };
```

(c) 発生条件(1)、(2)、(3)(C)、および(4)を満たしている場合

以下の方法で回避してください。

- 構造体のサイズが5バイト以上になるようにメンバを追加する。

回避例:

```
-----------------------------------------------------------------
 struct _str {
    __evenaccess long a;
    char dummy;          //構造体サイズが5バイト以上になるように
                  //メンバを追加する
 } ;
```

(d) 発生条件(1)、(2)、(3)(D)、および(4)を満たしている場合

以下のいずれかの方法で回避してください。

(d-1) D-3-1を満たしている場合

- ビットフィールドのビットサイズを2以上にする。

(d-2) D-3-2を満たしている場合

- ビットフィールドのビットサイズを2以上にする。または
  構造体のサイズが5バイト以上になるようにメンバを追加する。

回避例:

```
-----------------------------------------------------------------
 struct ST{
  __evenaccess unsigned long DATA:2;
```

```
}st;
```

2.2 ライブラリ関数 scanf()、 fscanf()、および sscanf()を使用する際の注意事項 (H8C-078)

該当バージョン:

V.4.0 ～ V.6.02 Release 00

現象:

scanf()、fscanf()、およびsscanf()の変換方式にs変換を使用すると、
空白(' ')、改行('¥n')、および水平タブ('¥t')を変換対象文字とみなして
変換し、引数ptrの指す記憶域へ格納します。

発生条件:

以下の条件をすべて満たしている場合に発生します。
(1) ライブラリ関数 scanf()、fscanf()、またはsscanf() を使用している。
(2) s変換を使用している。
(3) s変換対象の文字列に空白(' ')、改行('¥n')、および水平タブ('¥t')の
いずれかが含まれている。

発生例:

```
----------------------------------------------------------------------
 void main( void )
 {
    char tmp[ 10 ];
    sscanf( "ABC DEF", "%9s", tmp );    //発生条件(1),(2)
    printf( "%s¥n", tmp );              //"ABC DEF"と出力される。
 }
```

回避策:

以下の方法で回避してください。
- s変換の代わりに[変換を用いて、空白(' ')、改行('¥n')、水平タブ('¥t')
を変換対象文字から除外する。

回避例:

```
----------------------------------------------------------------------
 void main( void )
 {
    char tmp[ 10 ];
    sscanf( "ABC DEF", "%9[^ ¥t¥n]", tmp );  //s変換の代わりに[変換を
                                              //使用し空白、改行、
                                              //水平タブを記述する
```

```
    printf( "%s¥n", tmp );                  //"ABC"と出力される
}
```

2.3 式の中で同一変数を複数回使用する場合の注意事項(H8C-079)

該当バージョン:

V.6.01 Release 00 ～ V.6.02 Release 00

現象:

式の中で同一変数を複数回使用すると正しい演算結果が得られないことがあります。

発生条件:

以下の条件をすべて満たした場合に、発生することがあります。

(1) CPUオプションに 2000N、2000A、2600N、2600A、H8SXN、H8SXM、H8SXA、H8SXX、AE5、またはRS4を使用している。
(例: コマンドラインでは -cpu=2000N)

(2) 上記オプションで 2000N、2000A、2600N、または2600Aを使用し、出力オブジェクト互換オプション -legacy=v4 を使用していない。

(3) 最適化ありオプション -optimize=1を使用している。

(4) 加算、減算、乗算、除算、論理積、論理和、または排他的論理和を含む式内の被演算子に同一変数を複数回使用している。

発生例:

```
----------------------------------------------------------------------
unsigned long func2(unsigned short a);
unsigned long func1(unsigned long m)
{
    unsigned long r=0,tl=0;
    unsigned short r0=0,r1=0,t=0;
    unsigned short m0=0,m1=0;
    r=func2(m>>16);
    r0=r>>16;
    m1=m;
    tl=r0*r0; //発生条件(4)r0*r0が誤った演算結果になる
    t=tl>>16;
    tl=((unsigned long)m1*t+0x0000)>>15;
    r-=(unsigned long)tl<<0;
```

```
    return r-1;
}
```

出力コード例:

```
----------------------------------------------------------------
_func1:
        STM.L       (ER2-ER3),@-SP
        MOV.L       ER0,ER3
        MOV.W       E3,R0
        BSR         _func2:8
        MOV.L       ER0,ER2
        MULXU.W     E0,ER1      ;ER1に値を設定せず乗算している
        MOV.W       E1,E0
        MOV.W       E0,R1
        EXTU.L      ER1
        MOV.W       R3,R0
        EXTU.L      ER0
        MULU.L      ER0,ER1
        SHLR.L      #15:5,ER1
        SUB.L       ER1,ER2
        DEC.L       #1,ER2
        MOV.L       ER2,ER0
        RTS/L       (ER2-ER3)
----------------------------------------------------------------
```

回避策:

以下のいずれかの方法で回避してください。

(1) 最適化なしオプション -optimize=0を使用する。

(2) 当該現象が発生する関数の直前に #pragma option nooptimize を宣言する。

2.4 境界調整数に1を選択した構造体または共用体のメンバへアクセスする際の注意事項(H8C-080)

該当バージョン:

V.6.01 Release 00 ～ V.6.02 Release 00

現象:

境界調整数に1を選択した構造体または共用体のメンバへのアクセスが1バイトアクセスになりません。

発生条件:

以下の条件をすべて満たした場合に発生します。

(1) CPUオプションに2000N、2000A、2600N、2600A、またはRS4を使用している。
(例: コマンドラインでは -cpu=2600A)

(2) 出力オブジェクト互換オプション -legacy=v4 を使用していない。

(3) 構造体または共用体の境界調整数に1(-pack=1 または #pragma pack 1)を使用している。

(4) 以下の(A)または(B)の条件を満たしている。
(A) 以下の(A-1)～(A-3)の条件をすべて満たしている。
(A-1) (3)の構造体または共用体のサイズは4バイトである。
(A-2) (A-1)の構造体または共用体に4バイトのメンバを定義している。
(A-3) (A-1)の構造体または共用体型の変数を局所変数として宣言している。
(B) 以下の(B-1)～(B-4)の条件をすべて満たしている。
(B-1) (3)の構造体または共用体型のポインタを宣言している。
(B-2) (B-1)で述べた構造体または共用体のメンバに整数型でサイズが2バイトまたは4バイトのビットフィールドを定義している。
(B-3) (B-1)で述べた構造体または共用体のメンバが奇数アドレスに割り付けられている。
(B-4) (B-1)で述べた構造体または共用体の領域を確保していない。

(5) (4)(A-2)または(4)(B-2)のメンバを参照している。

発生例1:

```
#pragma pack 1                  //発生条件(3)
struct _str {
   long a ;                  //発生条件(4)(A-1),(4)(A-2)
} ;
void func(long p)
{
   volatile struct _str lstr ; //発生条件(4)(A-3)
   p = lstr.a ;               //発生条件(5)
}
```

出力コード:

```
   _func    SUBS.L    #4,ER7
            MOV.L     @ER7,ER0 ; 1バイトでアクセスしていない
```

発生例2:

```
-----------------------------------------------------------------
 #pragma pack 1 //発生条件(3)
 struct B {
    unsigned short US1:16; //発生条件(4)(B-2)
 };
 #define OBJB (*(volatile struct B*)0xFFFF23)
                        //発生条件(4)(B-1),(4)(B-3)
 void main(){
    OBJB.US1 = 1;          //発生条件(5)
 }
```

出力コード:

```
-----------------------------------------------------------------
 _main  MOV.W  #H'0001,R1
       MOV.W  R1,@H'00FFFF23:16  ; 1バイトでアクセスしていない
       RTS
```

回避策:

(1) 発生条件(1)、(2)、(3)、(4)(A)、および(5)を満たしている場合は、
以下の方法で回避してください。
構造体または共用体のサイズが4バイト以下の場合は5バイト以上に
なるようにダミーのメンバを追加する。

回避例:

```
-----------------------------------------------------------------
   struct _str {    //構造体のサイズを 5バイト以上にする
      long a ;
      char dummy;  //ダミーのメンバ
   }
```

(2) 発生条件(1)、(2)、(3)、(4)(B)、および(5)を満たしている場合は、
以下の方法で回避してください。
#pragma addressを使用して変数を割り付ける。

回避例:

```
-----------------------------------------------------------------
   #pragma address(OBJB=0xFFFF23)
   #pragma pack 1
   struct B {
      unsigned short US1:16;
   };
```

```
struct B OBJB;
void main(){
   OBJB.US1 = 1;
}
```

2.5 空ループ削除オプション del_vacant_loop=1 使用時に多重ループをコーディングする際の注意事項(H8C-081)

該当バージョン:

V.6.00 Release 00 ～ V.6.02 Release 00

現象:

空ではないループが削除されることがあります。

発生条件:

以下の条件をすべて満たした場合に発生することがあります。

(1) CPUオプションに2000N、2000A、2600N、2600A、H8SXN、H8SXM、H8SXA、H8SXX、H8SXX、AE5、またはRS4を使用している。
(例: コマンドラインでは -pu=2000N)

(2) 上記オプションで 2000N、2000A、2600N、または2600A を使用し出力オブジェクト互換オプション -legacy=v4を使用していない。

(3) 最適化オプションあり -optimize=1と使用している。

(4) 空ループ削除オプション -del_vacant_loop=1を使用している。

(5) 多重ループがある。

(6) 外側ループのループ制御式には、ループ終了判定文以外にもカンマ演算子で区切られた別の文がある。

(7) (6)のループおよび(6)のループより内側にあるすべてのループ内にはループ制御変数の更新式以外に式がない。

(8) (6)のループおよび(6)のループより内側にあるすべてのループのループ制御変数の値がループを抜けた後で参照されていない。

(9) V.6.00 Release 00～6.01 Release 02の場合で、(6)に該当するループ終了判定文以外の式で値が更新されている変数の型が、外側ループのループ制御変数の型と異なる。

発生例:

```
//-cpu=H8SXA:24
// while文の場合
// -opt=1 -del_vacant_loop=1
int a, b;
void func_while() {
  int i, j = 0;
  while (a++, b+=2, j < 4){       //発生条件(5),(6)
    for (i = 0; i < 500; i++) {   //発生条件(7)
    }
    j++;                          //発生条件(7)
  }
   //発生条件(8)
}

//-cpu=H8SXA:24
// for文の場合
// -opt=1 -del_vacant_loop=1
int c, d;
void func_loop() {
  int i, j;
  for (j = 0; c++, d+=2, j < 4; j++) {   //発生条件(5),(6),(7)
    for (i = 0; i < 500; i++) {          //発生条件(7)
    }
  }
  //発生条件(8)
}

// V.6.00 Release 00 ～V.6.01 Release 02
// for文の場合
// -opt=1 -del_vacant_loop=1
long c, d;
void func_loop() {
  int i, j;
  for (j = 0; c++, d+=2, j < 4; j++) {   //発生条件(5),(6),(7),(9)
    for (i = 0; i < 500; i++) {          //発生条件(7)
    }
  }
  //発生条件(8)
}
```

出力コード例:

```
-----------------------------------------------------------------
 _func_while
        RTS ;変数a,bの更新を行っていない
 _func_loop
        RTS ;変数c,dの更新を行っていない
-----------------------------------------------------------------
```

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)のループより内側にある、ループ内に処理のないループを
　コーディングしない。
(2) 空ループ削除オプションなし -del_vacant_loop=0を使用する。
(3) 最適化なしオプション -optimize=0を使用する。
(4) 当該現象が発生する関数の直前に #pragma option nooptimize を宣言
　する。

2.6 同一処理を複数ブロックに記述する際の注意事項(H8C-082)

該当バージョン:

V.6.01 Release 00 ～ V.6.02 Release 00

現象:

同一処理が複数ブロック*にあるときに、レジスタの値を設定しないで
レジスタを使用することがあります。
*'{'と'}'で囲まれたプログラム範囲

発生条件:

以下の条件をすべて満たした場合に、発生することがあります。
(1) CPUオプションに300、300L、300HN、300HA、2000N、2000A、2600N、
　2600A、H8SXN、H8SXM、H8SXA、H8SXX、AE5、またはRS4を使用している。
　(例: コマンドラインでは -cpu=300)
(2) 最適化ありオプション -optimize=1を使用している。
(3) 複数ブロックに同一処理がある。

発生例:

```
-----------------------------------------------------------------
 unsigned char chk0(void){}
 unsigned char chk1(void){}
 void sub(unsigned char byte){}
 void test(void)
 {
    if(chk0() == 1 || chk1() == 0){
```

```
        sub(0);    //発生条件(3)
    }else{
        sub(0);    //発生条件(3)
    }              //then節とelse節が同一処理になっている
}
```

出力コード例:

```
_test    BSR      @_chk0:8
         CMP.B    #H'01,R0L
         BEQ      @H'000E:8
         BSR      @_chk1:8
         BRA      @_sub:8   ;subのパラメータを設定していない
```

回避策:

以下のいずれかの方法で回避してください。
(1) 最適化なしオプション -optimize=0を使用する
(2) 当該関数の直前で#pragma option nooptimizeを宣言する。
(3) ブロック内にダミー関数の呼び出しか、ダミーの式を挿入する。

回避例:

```
//組み込み関数nop()の挿入する
#include <machine.h>
void test(void)
{
    if(chk0() == 1 || chk1() == 0){
        nop(); //組み込み関数nop()の呼び出し
        sub(0);
    }else{
        sub(0);
    }
}

//ダミーの式を挿入する
int dummy;
void test(void)
{
    if(chk0() == 1 || chk1() == 0){
        dummy = 0; //ダミーの式を挿入する
        sub(0);
    }else{
        sub(0);
```

```
    }
}
```

2.7 関数の戻り値を構造体、共用体、またはクラスのメンバ変数に代入する際の注意事項(H8C-083)

該当バージョン:

V.6.00 Release 00 ～ V.6.02 Release 00

現象:

構造体、共用体、またはクラス型のメンバに関数の戻り値を正しく代入しない
ことがあります。

発生条件:

以下の条件をすべて満たした場合に、発生することがあります。

(1) CPUオプションに2000N、2000A、2600N、2600A、H8SXN、H8SXM、H8SXA、
H8SXX、AE5、またはRS4を使用している。
(例: コマンドラインでは -cpu=2000N)

(2) 上記オプションで2000N、2000A、2600N、または2600A を使用し、
出力オブジェクト互換オプション -legacy=v4を使用していない。

(3) 最適化オプションあり -optimize=1を使用している。

(4) 戻り値の型のサイズが4バイト以下の関数呼び出しがある。

(5) (4)の戻り値の代入先は、サイズが4バイト以下の整数型または
単精度浮動小数点型の構造体メンバである。

(6) (5)のメンバを持つ構造体のサイズは、4バイト以下である。

(7) (6)の構造体型変数を局所変数として定義している。

発生例:

```
------------------------------------------------------------
//-cpu=h8sxa
struct ST{            //発生条件(6)
   int mem;
}str;
int y;
int sub(void);        //発生条件(4)
```

```
void temp2(){
   struct ST tmp2;    //発生条件(7)
   if (y == 0) {
     tmp2.mem = 3;
   } else {
     tmp2.mem = sub();//発生条件(4),(5)
   }
   sub2(tmp2.mem);
}
```

出力コード例:

```
-------------------------------------------------------------
_temp2:
        subs    #2,sp
        mov.w   @_y:32,r0
        bne     P_0000000e:8
        bra/s   P_00000014:8
        mov.w   #3:3,r0
P_0000000e:
        jsr     @_sub:24
        mov.w   @sp,r0 ;subの戻り値を上書きしている
P_00000014:
        jsr     @_sub2:24
        inc.l   #2,sp
        rts

-------------------------------------------------------------
```

回避策:

以下の(1)～(4)のいずれかの方法で回避してください。

(1) 構造体型の局所変数定義をvolatile修飾する。

```
-------------------------------------------------------------
//回避例:
void temp2(){
  volatile struct ST tmp2;
```

(2) 構造体型の局所変数定義を大域変数定義にする。

```
-------------------------------------------------------------
//回避例:
struct ST tmp2; //局所変数定義を大域変数にする
void temp2(){
   if (y == 0) {
```

(3) 最適化なしオプション -optimize=0を使用する。

(4) 当該関数の直前に#pragma option nooptimizeを宣言する。

```
------------------------------------------------------------
//回避例:
#pragma option nooptimze
void temp2(){
  struct ST tmp2;
. . . . . .
   sub2(tmp2.mem);
}
#pragma option
```

3. 内容(アセンブラ)

3.1 jmp命令の分岐先のアドレスに関する注意事項(H8A-0001)

該当バージョン:

V.4.0 ～ V.6.02 Release 00

現象:

jmp命令のオペランドである分岐先アドレスに間違った値を設定し正しく動作しないことがあります。

発生条件:

以下の(A)または(B)の条件を満たした場合に発生します。

(A) 以下の条件をすべて満たしている。
- (A-1) アセンブラ実行時に最適化オプション -optimizeを使用している。
- (A-2) プログラム領域のセクションを相対アドレス形式で宣言している。
  (以後、相対セクションと記載)
- (A-3) (A-2)のセクション内に最適化の結果、最大コードサイズとならない命令がある。*
  * 最適化の結果、コードサイズが変更される命令
- (A-4) (A-3)の命令の後方(アドレスの昇順の方向)にローカルラベルがある。
- (A-5) (A-4)のローカルラベルを参照する命令がある。

(B) 以下の条件をすべて満たしている。
- (B-1) プログラム領域のセクションを絶対アドレス形式で宣言している。
  (以後、絶対セクションと記載)
- (B-2) (B-1)のセクション内にローカルラベルがある。
- (B-3) 絶対セクション内で、オペランドにローカルラベルとアドレス未解決シンボルとのアドレス演算の書かれている命令がある。

発生例1:

```
------------------------------------------------------------
```

```
.section sec,code            ;発生条件(A-2)
      mov.l   #EQU1,@ER0     ;発生条件(A-3)
?LOCAL1                      ;発生条件(A-4)
      nop
      mov.l   #?LOCAL1,ER0   ;発生条件(A-5)
EQU1:  .equ    4
.end
```

発生例2:

```
.section sec,code,locate=H'100        ;発生条件(B-1)
.import imp_sym
      mov.l #imp_sym + ?LOCAL1, ER0   ;発生条件(B-3)
?LOCAL1                               ;発生条件(B-2)
      nop
.end
```

回避策:

以下の(A)または(B)の記載項目のいずれかの方法で回避してください。

(A) 発生条件(A)を満たしている場合
  (1) 最適化オプション -optimizeを使用しない。
  (2) ローカルラベルを使用しない。
  (3) 最適化によってコードサイズが変更される命令の第1オペランドを、
      定数のオペランドに変更するか(回避例1)、または同オペランドに
      確保サイズを記載する(回避例2)。

回避例1:

```
.section sec,code
      mov.l   #4,@ER0         ;定数に変更する
?LOCAL1
      nop
      mov.l   #?LOCAL1,ER0
EQU1:  .equ    4
.end
```

回避例2:

```
------------------------------------------------------------
.section sec,code
      mov.l   #EQU1:8,@ER0    ;確保サイズを記述する
?LOCAL1
      nop
      mov.l   #?LOCAL1,ER0
EQU1:  .equ    4
.end
------------------------------------------------------------
```

(B) 発生条件(B)を満たしている場合:
  (1) ローカルラベルを使用しない。
  (2) セクションを相対アドレス形式で宣言する。

3.2 絶対セクション内のラベルの記載に関する注意事項(H8A-0002)

該当バージョン:
V.4.0 ～ V.6.02 Release 00

現象:
絶対セクションのラベルを条件分岐命令のオペランドに使用した
場合に、最適化により誤ったアドレスに分岐します。*
* セクションを絶対アドレス形式で宣言している

発生条件:
以下の条件を全て満たした場合に発生します。

(1) 最適化オプション -optimizeを使用している。
(2) 絶対セクション内にラベルの記載がある。
(3) (2)のラベルを相対セクション内の条件分岐命令で参照している。
(4) (2)のラベルより前方(アドレスの降順の方向)に、最適化によって
    最大コードサイズとならない命令がある。*
    * 最適化の結果、コードサイズが変更される命令

発生例:

```
------------------------------------------------------------
.section rel_sec,code
      bra     ABS_LAB             ;発生条件(3)
      nop
.section abs_sec,code,locate=H'200
      mov.l   @(EQU1,ER1),ER0      ;発生条件(4)
ABS_LAB:                          ;発生条件(2)
      nop
```

```
EQU1:  .equ    4
.end
-------------------------------------------------------------
```

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を使用しない。

(2) 最適化によってコードサイズが変更される命令の第1オペランドを、
定数のオペランドに変更する(回避例1)。

(3) 最適化によってコードサイズが変更される命令の第1オペランドに
確保サイズを記述する(回避例2)。

(4) 条件分岐命令で参照しているラベルのセクションを相対アドレス形式
で宣言する。

回避例1:

```
-------------------------------------------------------------
 .section rel_sec,code
       bra     ABS_LAB
       nop
 .section abs_sec,code,locate=H'200
       mov.l   @(4,ER1),ER0    ;定数に変更する
       ABS_LAB
       nop
 EQU1:  .equ    4
 .end
-------------------------------------------------------------
```

回避例2:

```
-------------------------------------------------------------
 .section rel_sec,code
       bra     ABS_LAB
       nop
 .section abs_sec,code,locate=H'200
       mov.l   @(EQU1:2,ER1),ER0    ;確保サイズを記述する
 ABS_LAB
       nop
 EQU1:  .equ    4
 .end
-------------------------------------------------------------
```

4. 内容(最適化リンケージエディタ)

4.1 引数格納レジスタ数を3で宣言したアセンブラルーチンを呼び出す際の注意事項 (LNK-0001)

該当バージョン:

V.4.0 ～ V.6.02 Release 00

現象:

引数格納レジスタ数を3で宣言したアセンブラルーチンに正しく引数を渡すことができないことがあります。

発生条件:

以下の条件を全て満たした場合に発生することがあります。

(1) コンパイルオプションに -goptimizeを使用している。

(2) 呼び出し側の関数が以下の(A)～(C)の全てを満たしている。
  (A) 呼び出し側の関数がC/C++ソース内で他の関数を呼び出している。
  (B) 呼び出し側の関数の引数を格納するレジスタは2本である。
  (C) 呼び出される側の関数の引数を格納するレジスタは3本である。

(3) 呼び出される側がルーチンの場合に以下の(A)の条件を満たしているか、
  呼び出される側が関数の場合に以下の(B)の条件を満たしている。

  (A) 以下の(A-1)および(A-2)の条件を満たしている。
    (A-1) 呼び出されるルーチンをアセンブラソース内で定義している。
    (A-2) 引数格納レジスタER2、R2、およびE2のいずれかを引数
      として参照している。

  (B) 以下の(B-1)および(B-2)の条件を満たしている。
    (B-1) 呼び出される関数をC/C++ソース内で定義している。
    (B-2) 呼び出される関数は別関数を呼び出し、それぞれが相互に呼び
      出す関数である。

(4) 以下のいずれかを満たすことにより、リンク時の最適化オプション
  -optimize=registerが有効になっている。
  (A) -nooptimizeを使用していない。
  (B) -optimize=register を使用している。
  (C) -optimize をサブオプションなしで使用している。
  (D) -optimize=safe を使用している。
  (E) -optimize=speed を使用している。

発生例1:

```
------------------------------------------------------------
--- Cソース a.c ----
//ch38 -cpu=h8sxa -goptimize a.c
extern void __regparam3 fasm(long,long,long);//発生条件(2)(C)
```

```
long dmy_val;
void jibun();
void fc1(long hiki_dmy){
   fc2();
   dmy_val=hiki_dmy;
}
void fc2(){
   fasm(1,2,3);                          //発生条件(2)(A)
}
--- アセンブラソース b.src ---
; asm38 -cpu=h8sxa b.src
   .CPU       H8SXA:24
   .EXPORT    _fasm
   .EXPORT    _val
   .SECTION   P,CODE,ALIGN=2
_fasm:                                   ;発生条件(3)(A-1)
   ADD.L      ER1,ER0
   ADD.L      ER2,ER0                    ;発生条件(3)(A-2)
   MOV.L      ER0,@_val:32
   RTS
   .SECTION   B,DATA,ALIGN=2
_val:
   .RES.L     1
   .END

--- リンク時コマンド ---
optlnk -optimize=register a.obj b.obj -start=P,B/400
-------------------------------------------------------------
```

発生例2:

```
-------------------------------------------------------------
--- Cソース a.c ----
//ch38 -cpu=h8sxa -goptimize a.c
long dmy,result;
char own(long );
void __regparam3 child(long,long,long ); //発生条件(2)(C)

void loop();

void root (long par ){
   own (par);
   dmy+=par;
}
```

```
char own (long par){
   child(1,2,3);                    //発生条件(2)(A)

   dmy = par;
   if(result != 3 )
      return -1;
   else
      return 0 ;
}

void __regparam3 child (long par1,long par2 , long par3){
//発生条件(3)(B-1)
   if(par3 != 3 ){
      loop();                       //発生条件(3)(B-2)
   }
   result = par3;
}

void loop (){
   child(0,0,0);                    //発生条件(3)(B-2)
}
--- リンク時コマンド ---
optlnk -optimize=register a.obj -start=P,B/400
```

回避策:

以下のいずれかの方法で回避してください。
(1) リンク時に最適化オプション -optmize=registerを使用しない。
(2) 該当するCソース(上記例では、a.c)のコンパイル時にモジュール間
　最適化オプション -goptimizeを使用しない。

4.2 短絶対アドレッシングモード活用最適化オプション使用時の変数初期値に関する注意事項(LNK-0002)

該当バージョン:

V.4.0 ～ V.6.02 Release 00

現象:

短絶対アドレッシングモード活用最適化オプション
-optimize=variable_access の選択時に、変数の初期値を正しく参照ができないことがあります。

発生条件:

以下の条件を全て満たした場合に発生することがあります。

(1) C/C++ソースをモジュール間最適化オプション -goptimizeを使用して
コンパイルしている。

(2) (1)のソース内に初期値を設定した変数またはconst宣言された変数が
ある。

(3) (2)の変数の型は整数型または単精度浮動小数点型である。

(4) (2)の変数の初期値に関数のアドレス、変数のアドレス、または
セクションアドレス演算子を使用したアドレスのいずれかを設定して
いる。

(5) リンク時にアドレス整合性チェックオプション -cpu を使用していて、
そこで設定されたROM領域と、abs16宣言されている変数領域が重複して
いる。

(6) 以下のいずれかを満たすことにより、リンク時の最適化オプション
-optimize=variable_accessが有効になっている。
(A) -nooptimizeを使用していない。
(B) -optimize=registerを使用している。
(C) -optimize をサブオプションなしで使用している。
(D) -optimize=speedを使用している。
(E) -optimize=variable_accessを使用している。

発生例:

```
------------------------------------------------------------
--- Cソース a.c ----
//ch38 -cpu=2600a -goptimize a.c
long val;
long val_addr = (long)&val;
char func(){
  if (val_addr == (short)&val )
  . . . . . .
}
 --- リンク時コマンド ---
optlnk -optimize=variable_access a.obj -cpu=ROM=0-7fff,ROM=10000-11000 -
start=P,D,B/10000
------------------------------------------------------------
```

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(2)の変数のラベルを最適化部分抑止オプション

-variable_fobidに使用してリンクする。
(2) リンク時に最適化オプション -optimize=variable_accessを使用しない。
(3) リンク時に最適化オプション -nooptimizeを使用する。
(4) 該当するCソース(上記例では、a.c)のコンパイル時にモジュール間
最適化オプション -goptimizeを使用しない。

4.3 共通コード統合最適化オプション使用に関する注意事項(LNK-0003)

該当バージョン:
V.4.0 ～ V.6.02 Release 00

現象:
共通コード統合最適化(-optimize=same_code)が有効なときに、文字列の参照、倍精度浮動小数点型変数への代入が正しく行われないことがあります。

発生条件:
以下の条件を全て満たした場合に発生することがあります。

(1) C/C++ソースをモジュール間最適化オプション -goptimizeを使用してコンパイルしている。

(2) (1)のソース内に文字列の参照または倍精度浮動小数点型変数への代入がある。または、コンパイラで生成されたコードの中に、変数アドレスと定数を加算してアドレスを参照する命令がある。

(3) 以下のいずれかを満たしている。
(A) 共通コードの統合最適化オプション -optimize=same_codeおよび短絶対アドレッシングモード活用最適化オプション -variable_accessを使用している。
(B) スピード重視最適化オプション -optimize=speedを使用している。
(C) -optimize をサブオプションなしで使用している。
(D) -nooptimizeを指定していない。

発生例:

```
------------------------------------------------------------
--- Cソース a.c ----
// ch38 -cpu=h8sxa -goptimize -op=0 tp1.c

short check( char* a, long dmy_para){
  if( *a ==’s’) return 0;
  return 0;
}
short func1(){
```

```
  char *p =”string”;
  return check(p,0xABCDEF);
}
short func2(){
  char *p=”string”;
  return check(p,0xABCDEF);
}
short func(){
  if( 0 == func1() ) return 0;
  return 1;
}
```

```
 --- リンク時コマンド ---
 optlnk -optimize=variable_access,same_code -cpu=ROM=0-7fff,ROM=10000-
11000 -start=P,D,B/10000 a.obj
 ----------------------------------------------------------
```

回避策:

以下のいずれかの方法で回避してください。

(1) リンク時最適化オプション -optimize=same_codeを使用しない。

(2) リンク時最適化オプション -nooptimizeを使用する。

(3) 該当するCソース(上記の例では、a.c)のコンパイル時にモジュール
間最適化オプション -goptimizeを使用しない。

5. 恒久対策

本内容は、V.6.02 Release 01で改修する予定です。