# RENESAS ツールニュース

**RENESAS TOOL NEWS 2006年11月01日：061101/tn1**

## H8SX, H8S およびH8ファミリ用 C/C++コンパイラパッケージ V4～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. **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 組み込み関数movfpeおよびmovtpeに関する注意事項(H8C-0045)**

   **該当バージョン:**
   V.6.00 Release 00 ～ V.6.00 Release 03,
   V.6.01 Release 00 ～ V.6.01 Release 02

   **現象:**
   組み込み関数movfpeおよびmovtpeが正しく実行されない場合があります。

**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2000N,2000A,2600N,2600A,H8SXN,H8SXM,H8SXA,
H8SXXまたはAE5を選択している (コマンドラインの例:-cpu=2000n) 。
ただし、以下のいずれかの場合を除く。
・V.6.00で、CPUオプションとして2000N,2000A,2600Nまたは2600Aを
選択している場合
・V.6.01で、出力オブジェクト互換オプションを
選択している場合(コマンドラインでは-legacy=v4)

(2) 組み込み関数movfpeまたはmovtpeを関数内で複数回呼び出している。

(3) 最適化オプションを選択している(コマンドラインでは-optimzize=1
(デフォルト値))。

**発生例1:**

```
-------------------------------------------------
 #include <machine.h>
 #pragma abs16 a
 char a, data;
 void main(void){
   movfpe(&a,data); //発生条件(2)
   movfpe(&a,data); //発生条件(2)
 }
-------------------------------------------------
```

**生成コード1:**

```
-------------------------------------------------
 movfpe.b   @_a:16,r0l   ;movfpeが一度しか実行されない
 mov.b      r0l,@_data:32
-------------------------------------------------
```

**発生例2:**

```
-------------------------------------------------
 #include <machine.h>
 #pragma abs16 a
 char a, data;
 void main(void){
   movfpe(&a,data); //発生条件(2)
   movtpe(data,&a); //発生条件(2)
 }
-------------------------------------------------
```

**生成コード2:**

```
-------------------------------------------------
 movfpe.b   @_a:16,r0l
 mov.b      r0l,@_data:32
```

;movtpeが実行されない
---

**回避策:**
以下のいずれかの方法で回避してください。
(1) movfpeおよびmovtpe の引数に渡す変数をvolatile修飾する。
例:
```
#include <machine.h>
#pragma abs16 a
volatile char a, data; //volatile修飾する
void main(void){
  movfpe(&a,data);
  movfpe(&a,data);
}
```
(2) 最適化オプションを選択しない(コマンドラインでは-optimzize=0)。
(3) 外部変数の最適化抑止オプションを選択する(コマンドラインでは
-volatile)。
(4) V.6.01でCPUオプションとして、2000N,2000A,2600Nまたは2600Aを
選択している場合は、出力オブジェクト互換オプションを選択する
(-legacy=v4)。

## 2.2 型変換を伴った演算に関する注意事項(H8C-0046)

**該当バージョン:**
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**
型を拡張して除算または剰余算を行い、演算結果を拡張前の型に型変換した
場合、その演算結果はオーバーフローになります。

**発生条件:**
以下のすべての条件を満たす場合に発生します。
(1) CPUオプションとして2000N,2000A,2600Nまたは2600Aを選択している。
(例:-cpu=2000n)
(2) 出力オブジェクト互換オプション(コマンドラインでは-legacy=v4)を
選択していない。
(3) Cソースプログラム内に、除算または剰余算が記述されている。
(4) (3)の除数と被除数がどちらともsigned int型またはsigned short型から
signed long型への型変換が行われている。
(5) (3)の演算結果をsigned int型またはsigned short型の変数に代入
している。

(6) (3)の除数と被除数の値が、それぞれ-1と-32768である。

**発生例:**

```
---------------------------------------------------------------------
 #include <stdio.h>
 void init(void);
 short a,b,c;
 void main(void){
   init();
   a  = ((long)b / (long)c); //発生条件(3),(4),(5)
 }

 void init(void){
   a = 0;
   b = -32768;              //発生条件(6)
   c = -1;
 }
---------------------------------------------------------------------
```

**生成コード:**

```
---------------------------------------------------------------------
_main:
  ...
  mov.w      @_b:32,r0
  exts.l     er0
  mov.w      @_c:32,r1
  divxs.w    r1,er0    ;オーバーフローする
  mov.w      r0,@_a:32

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

**回避策:**

以下のいずれかの方法で回避してください。

(1) 出力オブジェクト互換オプションを選択する(-legacy=v4)。

(2) 除算または剰余算の除数と被除数のいずれかを、volatile修飾した
 signed long型変数に代入後、演算を行う。

例:

```
 ---------------------------------------------------------------------
 #include <stdio.h>
 short a,b,c;
 void main(void){
   volatile long lb; //volatile修飾する
   init();
```

```
  lb = (long)b;
  a  = (lb / (long)c);
}

void init(void){
  a = 0;
  b = -32768;
  c = -1;
}
```
---

## 2.3 ライブラリ関数strcpyおよびmemcpyのインライン展開に関する注意事項 (H8C-0047)

**該当バージョン:**
V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**
ライブラリ関数strcpyまたはmemcpyを使用し、かつ
インライン関数指定オプションを使用したとき(コマンドラインでは
library=intrinsic)に、プログラムが正しく動作しません。

**発生条件:**
以下のすべての条件を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を選択している。
  (例:-cpu=h8sxn)
(2) 特定ライブラリ関数のインライン関数指定オプションを選択している
  (-library=intrinsic)。
(3) ライブラリ関数strcpyまたはmemcpyを使用している。
(4) 以下のいずれかの条件を満たす。
  (a) E4以外のレジスタを#pragma global_registerの対象として
    選択している。
  (b) memcpyを使用している場合、memcpyに第3引数として0x60001以上
    の定数、または任意の定数アドレスを渡している。

発生例1:
```
#include <string.h>
char *dst;
void test(void){
  strcpy(dst, (char *)0); //発生条件(3),(4)-(b)
}
```

```
-------------------------------------------------
生成コード1:
-------------------------------------------------
 _dst:      ;strcpyのコードを生成していない
 .RES.L     1
-------------------------------------------------
```

```
発生例2:
-------------------------------------------------
 #include <string.h>
 char *dst;
 void test(void){
   strcpy(dst, (char *)10); //発生条件(3),(4)-(b)
 }
-------------------------------------------------
生成コード2:
-------------------------------------------------
 mov.w      #h'000a:16,r4;転送サイズが10バイト
 mov.l      @_dst:32,er6
 mov.l      #h'000a:16,er5
 movsd.b    ($+4)
 rts/l      (er4-er6)
-------------------------------------------------
```

**回避策:**

以下の方法で回避してください。

(1) 特定ライブラリ関数のインライン関数指定オプション
(コマンドラインでは-library=intrinsic)を選択しない。

## 2.4 #pragma inline_asmに関する注意事項(H8C-0048)

**該当バージョン:**

V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**

#pragma inline_asm宣言された関数内でcaller-saveレジスタ*1(R0L, R0H, R0, E0, ER0, R1L, R1H, R1, E1, ER1)のいずれかを使用しているときに、間違った実行結果になる場合があります。

*1:詳細はユーザーズマニュアルの「9.3.2(3) レジスタに関する規則」を参照してください。

**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2000N,2000A,2600N,2600A,H8SXN,H8SXM,H8SXA,
H8SXXまたはAE5を選択している。(例:-cpu=2000n)
ただし、以下のいずれかの場合を除く。
・V.6.00で、CPUオプションとして2000N,2000A,2600Nまたは2600Aを
選択している場合
・V.6.01で、出力オブジェクト互換オプションを選択している場合
(-legacy=v4)
(2) オブジェクト形式オプションのアセンブリプログラムを選択している
(-code=asmcode)。
(3) #pragma inline_asmで宣言された関数がある。
(4) (3)の関数内でR0L, R0H, R0, E0, ER0, R1L, R1H, R1, E1, ER1の
レジスタに対して、値を設定または更新を行う命令を使用している。
(5) (3)の関数を呼び出す式がある。
(6) (3)の関数は、戻り値がないか、または(5)の呼び出し式でその戻り値
を参照していない。
(7) 以下の(7)-1の条件、または(7)-2のすべての条件、のいずれかを
満たしている。
(7)-1
(5)の式を含む関数では、(3)の関数以外の関数を呼び出す式がない。
(7)-2
(a) #pragma interruptまたは、__interrupt宣言された関数がある。
(b) (a)の関数は、#pragma regsaveまたは、__regsave宣言
されていない。
(c) (a)の関数内で呼び出されている関数は全てインライン展開
されている。

**発生例:**

```
----------------------------------------------------
 #pragma inline_asm g  //発生条件(3)
 static void g(void){  //発生条件(6)
   MOV.L  #0,ER1     //発生条件(4)
 }
 int f(int a,int b) {
   g();              //発生条件(5)
   return a+b;
 }
----------------------------------------------------
```

**生成コード:**

```
----------------------------------------------------
 _f:
 mov.l       er0,er1 ;引数a,bの値をer1へコピー
 mov.l       #0,er1  ;er1の値を上書き
```

```
 add.w       e1,r1
 mov.w       r1,r0
 rts
```
---

**回避策:**
以下のいずれかの方法で回避してください。

(1) __asm と #pragma inline を使用する。
例:
```
 #pragma inline (g)
 static void g(void) {
   __asm{
      MOV.L  #0,ER1
   }
 }
 int f(int a,int b) {
   g();
   return a+b;
 }
```
---

(2) __asm を使用してアセンブリコードを記述する。
例:
```
 int f(int a,int b) {
   __asm{
      MOV.L  #0,ER1     //inline_asm指定した関数のコードを記述する。
   }
   return a+b;
 }
```
---

(3) #pragma interrupt 宣言をした関数に対し、レジスタ退避/回復コード
制御機能 #pragma regsave、または __regsave を宣言する。
(発生条件(7)-2に該当している場合のみ)
例:
```
 #pragma regsave(func)
 #pragma interrupt(func)
```
---

(4) インライン展開されないダミーの関数を定義し、#pragma interrupt
を宣言した関数からダミーの関数を呼び出す。
(発生条件(7)-2に該当している場合のみ)

例:

```
-------------------------------------------------
 void dummy();
 void func(void){
   sub();          //inline_asm 宣言された関数の呼出し
   dummy();        //ダミー関数を呼び出す
 }
 void dummy(){}
-------------------------------------------------
```

## 2.5 定数式の添え字に関する注意事項(H8C-0049)

**該当バージョン:**

V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**

配列の添え字に、式の評価結果が定数式となる式を記述したとき、
間違った領域をアクセスするコードを生成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
選択している。(例:-cpu=h8sxn)
(2) 配列の添え字に式を記述している。
(3) (2)の式を評価した結果が、定数式になる。

**発生例:**

```
-------------------------------------------------
 struct {
   int pad;
   int aaa;
 } a[20];
 int x;

 void main(){
   a[(x*0)+3].aaa = 20;//発生条件(2),(3)
 }
-------------------------------------------------
```

**生成コード:**

```
--------------------------------------------------
 _main:
 mov.w      #5:3,r0  ;定数値3をレジスタr0に代入しなくては
                 いけないが、間違って5が代入される
 mova/l.l   @((_a+2):32,r0.w),er1
 mov.w      #h'0014:16,r0
 mov.w      r0,@er1
--------------------------------------------------
```

**回避策:**

以下のいずれかの方法で回避してください。

(1) 配列の添え字式をvolatile修飾した変数に代入してから、
配列にアクセスする。

例:

```
--------------------------------------------------
 struct {
   int pad;
   int aaa;
 } a[20];
 int x;
 volatile unsigned long ul;//volatile修飾した変数を定義

 void main(){
   ul = (x*0)+3;//volatile修飾した変数へ代入
   a[ul].aaa = 20;
 }
--------------------------------------------------
```

(2) 配列の添え字を定数にする。

例:

```
--------------------------------------------------
 struct {
   int pad;
   int aaa;
 } a[20];
 int x;

 void main(){
   a[3].aaa = 20;//添え字を定数で記述
 }
--------------------------------------------------
```

## 2.6 サイズが3バイトの構造体型または共用体型の変数またはメンバの代入に関する注意事項(H8C-0050)

**該当バージョン:**
V.4.0～V.4.0.09,
V.5.0～V.5.0.06,
V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**
メモリまたはスタックに割り付いた、サイズが3バイトの
構造体型または共用体型の変数またはメンバをレジスタにコピーすると、
レジスタに割り付いている1バイトの変数の値が間違った値になる。

**発生条件:**
以下の(1)-1のすべての条件、または(1)-2のすべての条件、のいずれかを
満たす場合に発生することがあります。
(1)-1
(a) コンパイラパッケージのバージョンがV.6.01 Release 00 ～
V.6.01 Release 02のいずれかである。
(b) CPUオプションとして2000N,2000A,2600Nまたは2600Aを選択している。
(例:-cpu=2000n)
(c) 出力オブジェクト互換オプション(コマンドラインでは-legacy=v4)を
選択していない。
(d) スピード優先最適化オプション(コマンドラインでは
-speed=expressionおよび-speed=struct)を選択していない。
(e) メモリまたはスタックに割り付いた、サイズが3バイトの構造体型
または共用体型のメンバ、またはサイズが3バイトの構造体型または
共用体型の変数がある。
(f) レジスタに割り付いた、サイズが3バイトの構造体型または共用体型の
メンバ、またはサイズが3バイトの構造体型または共用体型の変数が
ある。
(g) (f)のメンバまたは変数が割り付いているレジスタの1バイトに
他の変数が割り付いている。
(h) (e)のメンバまたは変数から、(f)のメンバまたは変数への代入式が
ある。
(i) (h)の代入の後に、(f)のメンバまたは変数への代入がある。

(1)-2
(a) CPUオプションとして300HN,300HA,2000N,2000A,2600Nまたは2600Aを
選択している。(例:-cpu=300hn)
(b) 出力オブジェクト互換オプションを選択している(-legacy=v4)。
(V.6.01の場合のみ)

(c) 局所変数（引数を含む）に構造体型または共用体型の変数を
使用している。
(d) (c)の変数は、サイズが4バイトでかつ、サイズが3バイトの
構造体型または共用体型のメンバが先頭メンバにある。
(e) (c)の変数がレジスタ割り付いている。

**発生例:**

```
----------------------------------------------------
typedef struct {
  char c[3];
}ST3;

typedef struct {
  ST3  st3;
  char c3;
}ST4;

ST3  m_st = { 0, 1, 2 };                  //発生条件(1)-1 (e)

ST4 sub(ST4  r_st){
  r_st.st3 = m_st;                        //発生条件(1)-1 (h)
  return(r_st);
}
void main(void){
  ST4  r_st = { { -1, -1, -1 }, -1 };     //発生条件(1)-1 (f),(g)

  r_st = sub(r_st);

  if(r_st.c3==-1){                        //発生条件(1)-1 (i)
  ...
  } else {
  ...
  }
}
----------------------------------------------------
```

**生成コード:**

```
----------------------------------------------------
_sub:
  push.l  er2
  mov.l   #_m_st:32,er0
  jsr     @$mv3mr$:24         ;3バイトをコピー
  mov.l   er0,er2
  pop.l   er2
```

```
    rts
_main:
    stm     (er2-er3),@-sp
    subs    #4,sp
    mov.b   @C_00000000:32,r2h
    mov.b   @C_00000001:32,r2l
    mov.w   r2,e3
    mov.b   @C_00000002:32,r3h
    mov.b   @C_00000003:32,r3l
    mov.l   er3,er0
    bsr     _sub:8
    mov.l   er0,er2          ;R2Lにある変数値を上書き
-------------------------------------------------
```

**回避策:**

(1) 発生条件(1)-1に該当する場合、以下のいずれかの方法で回避して
ください。

(a) スピード優先最適化オプションを選択する(-speed=expressionまたは
-speed=struct)。

(b) #pragma option speed=structまたは#pragma option speed=expression
を発生条件(1)-1の(h)の代入式を含む関数に指定する。

例:

```
-------------------------------------------------
 ...
 #pragma option speed=struct

 ST4 sub(ST4  r_st){
     r_st.st3 = m_st;
     return(r_st);
 }
 #pragma option
 ...
-------------------------------------------------
```

(2) 発生条件(1)-2に該当する場合、以下のいずれかの方法で回避して
ください。

(a) 発生条件(1)-2の(c)に該当する局所変数または引数に、volatile修飾子
を付加する。

(b) 構造体型または共用体型のメンバ変数の宣言する順序を変更する。

例:

```
-------------------------------------------------
  typedef struct {
```

```
    char c3;
    ST3  st3;
  }ST4;
 ------------------------------------------------
```

(c) 1バイト以上のダミーのメンバ変数を追加して、発生条件(1)-2の(c)の
構造体型または共用体型の変数のサイズを5バイト以上にする。

例:

```
 ------------------------------------------------
 typedef struct {
    ST3  st3;
    char c3;
    char dummy;      /* 使用しないメンバ変数を追加し、
                        構造体変数のサイズを5バイトにする。 */
 }ST4;
 ------------------------------------------------
```

## 2.7 #pragma option speed=register使用時の注意事項(H8C-0051)

**該当バージョン:**

V.4.0～V.4.0.09,
V.5.0～V.5.0.06,
V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**

#pragma option speedのサブオプションにregisterを
使用したとき、間違ったスタック領域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します。
(1) CPUオプションとして300,300HNまたは300HAを選択している。
(例:-cpu=300)
(2) 最適化オプションを選択している(-optimzize=1)。
(3) #pragma option speedを指定している、または#pragma option speedの
サブオプションにregisterを指定している
(#pragma option speed=register)。
(4) 保証されるレジスタ*1を使用している。使用された結果、そのレジスタの
退避および回復コードが出力されている。
(5) スタックに割りついた引数へのアクセスが発生している。

*1:詳細はユーザーズマニュアルの「9.3.2(3) レジスタに関する規則」を
参照してください。

**発生例:**

```
------------------------------------------------------------------------
typedef struct{
  short int_a;
  short int_b;
}union_data;
volatile static union_data data[2];

#pragma option speed=register              //発生条件(3)
void function(union_data* temp,long long_data,char index){
  short int_a;
  short int_b;

  int_a = temp->int_a;                  //ER6を使用
  int_b = temp->int_b;

  data[index].int_a = int_a * long_data; //発生条件(5)
  data[index].int_b = int_b * long_data* 16384;
}
------------------------------------------------------------------------
```

**生成コード:**

```
----------------------------------------------------
_function:                          ; function: function
    push.l      er6
    push.l      er4
    mov.l       er0,er6
    mov.w         @er6,r4
    mov.w        @(2:16,er6),e4
    mov.b        @(25:16,sp),r6l ;間違ったスタック領域をアクセス
    ...
----------------------------------------------------
```

**回避策:**

以下のいずれかの方法で回避してください。

(1) 最適化オプションを選択しない(-optimzize=0)。

(2) #pragma option speed および #pragma option speedのregisterサブオプションを使用せず、スピード優先最適化オプションを選択する(-speed=expressionまたは -speed=struct)。

(3) #pragma option speedのregisterサブオプションを使用しない。
#pragma option speed選択時は、
#pragma option speed=shift,loop,switch,inline,struct,expression
と記述する。

## 2.8 2つの1ビットのビットフィールドメンバを比較するときの注意事項 (H8C-0052)

**該当バージョン:**
V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**
2つの構造体または共用体の1ビットで指定されたビットフィールドの
メンバを比較したとき、間違った比較結果になる場合があります。

**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2000N,2000A,2600N,2600A,H8SXN,H8SXM,H8SXA,
H8SXXまたはAE5を選択している。(例:-cpu=2000n)
ただし、以下のいずれかの場合を除く。
・V.6.00で、CPUオプションとして2000N,2000A,2600Nまたは2600Aを
選択している場合
・V.6.01で、出力オブジェクト互換オプションを
選択している場合(-legacy=v4)
(2) 1ビットのビットフィールドメンバを含む構造体または共用体を、
宣言および定義している。
(3) ビットフィールドのサイズが1ビットのメンバ同士を比較している
式がある。
(4) (3)の比較は、等価演算子(!=または==)で比較している。

**発生例:**
```
--------------------------------------------------
 struct ST{
   int bit:1;
 };

 int x;
 struct ST st1;
 struct ST st2[50];
 void temp(){
   if (st1.bit == st2[x].bit) { //発生条件(3),(4)
       sub();
   }
 }
--------------------------------------------------
```
**生成コード:**

```
-------------------------------------------------
_temp:
   bld.b      #7,@_st1:32 ;キャリーフラグが立つ
   mov.w      @_x:32,r0
   exts.l     er0
   shll.l     er0         ;キャリーフラグを上書き
   mov.b      @(_st2:32,er0),r1l
   bxor.b     #7,r1l
   bcs        L28:8
-------------------------------------------------
```

**回避策:**

以下のいずれかの方法で回避してください。

(1) ビットフィールドをメンバに持つ構造体変数または共用体変数の
アドレスをポインタに代入してポインタを用いてアクセスする。

例:

```
-------------------------------------------------
struct ST{
   int bit:1;
};

struct ST *p1, *p2; //ポインタ変数を宣言
int x;
struct ST st1;
struct ST st2[50];
void temp(){
   p1 = &st1;
   p2 = &st2[x];
   if( p1->bit == p2->bit) {
      sub();
   }
}
-------------------------------------------------
```

(2) volatile修飾された同型変数に代入してから、比較を行う。

例:

```
-------------------------------------------------
struct ST{
   int bit:1;
};

int x;
```

```
 struct ST st1;
 struct ST st2[50];
 void temp(){
    volatile int a, b; //volatile修飾して変数宣言する
    a = st1.bit;
    b = st2[x].bit;
    if( a == b) {
       sub();
    }
}
```
---

## 2.9 繰り返し文に関する注意事項(H8C-0053)

**該当バージョン:**
V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**
繰り返し文内で実行されないプログラムフロー上に、例外を発生させる
可能性のある式があるときに、その式を実行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2000N,2000A,2600N,2600A,H8SXN,H8SXM,H8SXA,
H8SXXまたはAE5を選択している。(例:-cpu=2000n)
ただし、以下のいずれかの場合を除く。
・V.6.00で、CPUオプションとして2000N,2000A,2600Nまたは2600Aを
選択している場合
・V.6.01で、出力オブジェクト互換オプションを
選択している場合(-legacy=v4)
(2) 最適化オプションを選択している(-optimzize=1)。
(3) 繰り返し文がある。
(4) (3)の繰り返し文内に、除算式または剰余算式がある。
(5) (4)の除算式または剰余算式は、以下のすべての条件を満たしている。
(a) 除数は、0または、値が0となる式
(b) (4)の繰り返し文で値が変化する変数を含まない式
(c) (4)の繰り返し文で実行されない式

**発生例:**
---
```
 int a[100];
 int W=1, X=0, Y, Z;
```

```
 void f()
 {
   int i;
   for (i=0;i<100;i++){     //発生条件(3),(5)(b)
      if (X > 0) {
          Y = Z / (W-1);   //発生条件(4),(5)(a),(5)(c)
      }
      a[i] =1 ;
   }
 }
```
---

**回避策:**
以下のいずれかの方法で回避してください。

(1) 除算式または、剰余算式に含まれる変数のいずれかをvolatile修飾する。
例:
```
 int a[100];
 int W=1, X=0, Z;
 volatile int Y; //volatile修飾する
 void f()
 {
   int i;
   for (i=0;i<100;i++){
      if (X > 0) {
          Y = Z / (W-1);
      }
      a[i] =1 ;
   }
 }
```
---

(2) 除算式または剰余算式を削除する。
例:
```
 int a[100];
 int W=1; X=0, Z;
 void f()
 {
   int i;
   for (i=0;i<100;i++){
     // 実行されない式を削除する
```

```
      a[i] =1 ;
    }
  }
```
---

(3) 最適化オプションを選択しない(-optimzize=0)。

## 2.10 関数の戻り値の構造体メンバに直接代入に関する注意事項(H8C-0054)

**該当バージョン:**
V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**
関数の戻り値を構造体メンバに直接代入すると、同一構造体内に
宣言されている構造体メンバが保持している値が書き変わってしまう
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します。
(1) CPUオプションとして2000N,2000A,2600N,2600A,H8SXN,H8SXM,H8SXA,
H8SXXまたはAE5を選択している。(例:-cpu=2000n)
ただし、以下のいずれかの場合を除く。
・V.6.00で、CPUオプションとして2000N,2000A,2600Nまたは2600Aを
選択している場合
・V.6.01で、出力オブジェクト互換オプションを
選択している場合(-legacy=v4)
(2) 最適化オプションを選択している(-optimzize=1)。
(3) サイズが4バイト以下の構造体型変数を宣言および定義している。
(4) (3)の構造体型変数は、局所変数として宣言している。
(5) (3)の構造体型変数またはそのメンバに、volatile修飾がされていない。
(6) 関数の戻り値を、(3)の構造体のメンバに設定している。

**発生例:**
```
---------------------------------------------------------------------
#include <stdio.h>
typedef unsigned char   UC;
typedef unsigned short  US;
typedef unsigned long   UL;

typedef union _UDWORD       //発生条件(3),(5)
{
  UL  DWORD;
```

```
  struct{
    US  h;
    US  l;
  } WORD;
  struct{
    UC   eh;
    UC   el;
    UC   rh;
    UC   rl;
  } BYTE;
} UDWORD, *pUDWORD;

volatile US abs16_val1;
extern UC setval(void);
void sub(US,UC *, UC *);

void func(void)
{
  US  rs_u16;
  UC   ro_u8 ;
  UDWORD  Size, Time;     //発生条件(4),(5)
  UC   dmy1, dmy2;

  Size.BYTE.eh = setval(); //発生条件(6)
  Size.BYTE.el = setval();
  Size.BYTE.rh = setval();
  Size.BYTE.rl = setval();
  Time.BYTE.eh = setval();
  Time.BYTE.el = setval();
  Time.BYTE.rh = setval();
  Time.BYTE.rl = setval();

  if ( !ro_u8 ){
    if( Time.DWORD == 0 ){
      abs16_val1 = 0x0100;
    }
  }
  sub(rs_u16, &dmy1, &dmy2);
  if( Time.DWORD == 0 ){
    abs16_val1 = 0x0100;
  }
}
```

---

**生成コード:**

```
------------------------------------------------------------------------
_func:                          ; function: func
   ...
   jsr       @er2
   mov.b     r0l,@(H'000B:16,sp)
   jsr       @er2
   mov.w     e1,r3        ;er1に構造体全体をロードしていない
   mov.b     r0l,r3h
   mov.w     r3,e1
   mov.l     er1,@(12:2,sp);構造体全体をスタックに退避
------------------------------------------------------------------------
```

**回避策:**

以下のいずれかの方法で回避してください。

(1) 最適化オプションを選択しない(-optimzize=0)。

(2) 該当する関数に対して、#pragma option nooptimize拡張機能を
選択する。

(3) 構造体型変数を、volatile修飾する。

(4) 構造体型変数のサイズを、4バイトより大きくする。

## 2.11 __asmに関する注意事項(H8C-0055)

**該当バージョン:**

V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**

__asmを使用したときに、存在しない命令を出力して、
実行エラーとなる場合があります。

**発生条件:**

以下のすべての条件を満たす場合に発生します。

(1) CPUオプションとして2000N,2000A,2600Nまたは、2600Aを選択している。
(例:-cpu=2000n)

(2) __asm 内に、jsrまたはjmp命令を記述している。

(3) (2)の命令の分岐先が、32ビット絶対アドレスで記述されている。

**発生例:**

```
------------------------------------------------------------------------
extern void sub1(void);
extern void sub2(void);
void func(void)
{
```

```
  __asm{
    jsr @sub1:32 //発生条件(2),(3)
    jmp @sub2:32 //発生条件(2),(3)
  }
}
```

---

**回避策:**

以下のいずれかの方法で回避してください。

(1) オブジェクト形式オプションのアセンブリプログラムを選択している
(-code=asmcode)。

(2) jsrおよびjmp命令の絶対アドレス指定を、24ビットで記述する。

---

```
extern void sub2(void);
void func(void)
{
  __asm{
    jsr @sub1:24
    jmp @sub2:24
  }
}
```

---

## 2.12 関数アドレスの初期値に関する注意事項(H8C-0056)

**該当バージョン:**

V.6.00 Release 00 ～ V.6.00 Release 03,
V.6.01 Release 00 ～ V.6.01 Release 02

**現象:**

関数のアドレスを任意のデータ型へのポインタ型にキャストして、
定数と加算または減算した結果を静的記憶域の変数の初期値に指定した場合、
間違った変数の値になります。

**発生条件:**

以下のすべての条件を満たす場合に発生します。

(1) CPUオプションとして2000N,2000A,2600N,2600A,H8SXN,H8SXM,H8SXA,
H8SXXまたはAE5を選択している。(例:-cpu=2000n)
ただし、以下のいずれかの場合を除く。
・V.6.00で、CPUオプションとして2000N,2000A,2600Nまたは2600Aを
選択している場合
・V.6.01で、出力オブジェクト互換オプションを
選択している場合(-legacy=v4)

(2) 関数シンボルまたはそれにアドレス取得演算子(&)を付けたものを、
　データ型へのポインタ型でキャストし、定数を加算または減算する。
(3) (2)の演算結果を初期値とする静的記憶域の変数を定義する。

**発生例:**

```
----------------------------------------------------------------------
void func();
char *p = (char*)func + 1;  //発生条件(2),(3)
----------------------------------------------------------------------
```

**生成コード:**

```
----------------------------------------------------------------------
_p:                             ; static: p
.DATA.L     _func ; 定数値H'00000001が加算されていない
----------------------------------------------------------------------
```

**回避策:**

以下の方法で回避してください。
(1) 別ファイルに関数名を任意のデータ型配列名として宣言し、オブジェクト
　形式オプションのアセンブリプログラムを選択する(-code=asmcode)。

```
----------------------------------------------------------------------
//別ファイル
extern int func[];
char *p = (char*)func + 1;
----------------------------------------------------------------------
```

3. **恒久対策**

本内容はV.6.01 Release 03で改修する予定です。

---