H8 C/C++コンパイラ Ver.6 使用上のご注意

H8C/C++コンパイラ Ver.6 には、以下に記載しました不具合があります。H8C/C++コンパイラ Ver.6 をご使用の場合には、ご注意いただきますようお願いいたします。
なお、本ページに記載されている不具合は、コンパイラ Ver.6.0.01 以降（コンパイラパッケージ Ver.6.0.01 以降）では、全て修正されています。製品をリビジョンアップしていただきますよう、お願いいたします。

製品型名	パッケージバージョン	コンパイラバージョン
PSO08CAS6-MWR	6.0.00	6.0.00
PSO08CAS6-SLR	6.0.00	6.0.00
PSO08CAS6-H7R	6.0.00	6.0.00

1. 符号無し変数に対する定数除算時オブジェクト不正

現象

cpuexpand オプション指定時に、unsigned int(unsigned short)または unsigned long の変数に対し、2 の n 乗の定数値で除算を実施した場合、不正なコードが生成される場合があります。

[例]
```
  unsigned int a;
  unsigned long b;
  void sub(void)
  {
    a = b / 2048; /* 下位 2byte で除算を行うためコード不正となります。 */
  }
```

〈不正なコード〉
```
_sub:
    MOV.W @_b+2:32,R0
    SHLR.W #11:5,R0
    MOV.W R0,@_a:32
    RTS
```

〈正しいコード〉
```
_sub:
    MOV.L @_b:32,ER0
    SHLR.L #11:5,ER0
    MOV.W R0,@_a:32
    RTS
```

発生条件

以下の(1)、(2)いずれかの条件を満たす場合、発生することがあります。

(1) 以下の条件をすべて満たす場合
  (a) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
  (b) cpuexpand オプションを指定している。
  (c) (unsigned char)(変数 / 定数) の式を記述している。
  (d) 変数が unsigned int または unsigned short 型で、定数が 0-255 の範囲に収まる 2 の n 乗の値である。

(2) 以下の条件をすべて満たす場合
  (a) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
  (b) cpuexpand オプションを指定している。

(c) (unsigned int)(変数 / 定数) または(unsigned short)(変数 / 定数) の式を記述している。
(d) 変数が unsigned long 型で、定数が 0-65535 の範囲に収まる 2 の n 乗の値である。

回避策

該当箇所が存在した場合、下のいずれかの方法で回避してください。
(1) 演算結果を一度被除数と同じ型の変数に代入した後に、左辺の変数に代入を行う。

```
[例]
  unsigned int a;
  unsigned long b;
  void sub(void)
  {
    unsigned long c;

    c = b / 2048;
    a = c;
  }
```

(2) 除数を volatile つきの変数に設定し、その変数を用いて除算を行う。

2. 構造体初期値不正

現象

ポインタ型・スカラ型の順で宣言された構造体メンバに初期値を指定する場合、ポインタ型のメンバにアドレス+オフセット、スカラ型メンバに定数値を指定すると、初期値データが出力されない場合があります。

```
[例]
  unsigned char data[2] = { 0x00, 0x11 };
  const struct st_sample {
  void *d1;
  unsigned long d3;
  }st1 = { (void *)(data+1),
    0x22222222 /* 本記述が、オブジェクトに出力されません */
  };
```

発生条件

以下の条件をすべて満たす場合、発生することがあります。
(1) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
(2) ポインタ型、スカラ型のメンバが連続して宣言された構造体が存在する。
(3) (2)のポインタ型メンバとスカラ型メンバは同じサイズである。
(4) (2)のポインタ型メンバの初期値がアドレス+定数である。
(5) (2)のスカラ型メンバの初期値が定数値である。
(6) 構造体が const 指定されている。

回避策

該当箇所が存在した場合、以下の方法で回避してください。

・発生条件(2)の各メンバの間にサイズが異なるダミーのメンバを挿入する。

```
[例]
  unsigned char data[2] = { 0x00, 0x11 };
  const struct st_sample {
    void *d1;
    unsigned int u;
    unsigned long d3;
  }st1 = { (void *)(data+1),
    0xff,
    0x22222222
  };
```

3. ビットフィールド設定不正

現象

特定のビットフィールドに定数値を設定した場合、マスク値が不正になる場合があります。

```
[例]
  struct ST{
  signed long offset:3;
  signed long data:20;
  }st;

  void func(void)
  {
    st.data = 0x000000ff; /* 定数値を代入した場合、
                             マスク値が不正になります。 */
  }
```

〈不正なコード〉

```
_func:
    MOV.L @_st:32,ER0
    AND.L #h'dfffb3df:32,ER0
    OR.L #h'0001fe00:32,ER0
    MOV.L ER0,@_st:32
    RTS
```

〈正しいコード〉

```
_func:
    MOV.L @_st:32,ER0
    AND.L #h'e00001ff:32,ER0
    OR.L #h'0001fe00:32,ER0
    MOV.L ER0,@_st:32
    RTS
```

発生条件

以下の条件をすべて満たす場合、発生することがあります。

(1) CPU 種別として H8SXN,H8SXM,H8SXA,H8SXX のいずれかを指定している。

(2) ビットフィールドに定数値を代入している。

(3) 代入するビットフィールドが以下のいずれかである。

  (a) 型:long ビット幅:20 ビットオフセット:3

  (b) 型:long ビット幅:15 ビットオフセット:5

  (c) 型:long ビット幅:9 ビットオフセット:17

  (d) 型:long ビット幅:9 ビットオフセット:8

(e) 型:long ビット幅:9 ビットオフセット:0

(f) 型:int/short ビット幅:9 ビットオフセット:2

回避策

該当箇所が存在した場合、以下の方法で回避してください。

・発生条件(3)のパターンに当てはまらないように、当該ビットフィールドの前にダミーのビットフィールドを挿入する。

```
[例]
struct ST{
  signed long offset:3;
  signed long dummy:1; /* dummy のビットフィールドを挿入する */
  signed long data:20;
}st;
```

4. 3byte 構造体使用時の不正値設定

現象

structreg オプション指定時に 3byte の構造体の戻り値を記述、または 3byte の配列・ポインタ型の構造体に値を設定した場合、その構造体の次の領域に不正な値を設定する場合があります。

```
[例]
  #pragma pack 1
  typedef struct {
    char a;
    int b;
  } ST;
  #pragma unpack

  ST st2[3];
  ST sub();

  void main(void){
    st2[1] = sub(); /* 次の領域( st2[2].a)に不正に値が設定されます。 */
  }
```

発生条件

以下の条件を全て満たす場合、発生することがあります。

(1) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。

(2) structreg オプションを指定している。

(3) 関数の戻り値に 3byte 構造体を指定している、または配列・ポインタ型の 3byte 構造体に対して設定を行っている。

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。

(1) structreg オプションを指定せずにコンパイルする。
(2) 3byte の構造体に 1byte のダミーメンバを追加し、4byte の構造体とする。
(3) 当該構造体に volatile 修飾子を指定する。

5. 関数呼び出しの不正削除

現象

判定文の後に関数呼び出しがあり、その関数呼び出しで処理が終了するような関数を記述した場合、関数呼び出しが不正に削除される場合があります。

[例]

```
  extern unsigned char a,b,array1[],array2[];
  void func01();
  void func02();
  void sub(void)
  {
    if(a==1) {
      if(array2[1]==0x01) {
        func02( ); /* func02 呼出し後に本関数での処理がないため、func02 関数 */
                   /* 呼び出しが不正に削除されます。 */
      } else {
        func01( );
      }
    } else if(a==2) {
      if(b&0x01) {
      func02( );
      } else {
        func01( );
      }
    } else {
      if(b&0x01) {
        array1[1]=0x08;
      } else {
        func01( );
      }
    }
  }
  void func01(){}
  void func02(){}
```

発生条件

以下の条件を全て満たす場合、発生することがあります。
(1) CPU 種別として 300, 300HN, 300HA, 2000N, 2000A, 2600N, 2600A のいずれかを指定している。
(2) goptimize オプションを指定していない。
(3) 呼び出し元関数入口/出口にレジスタの退避/回復がない。
(4) (3)の関数内において、判定文の then 節の直後に関数呼び出しがある。
(5) (4)の呼び出される関数は以下を満たしている。
  (a) 引数渡しにスタックを使用しない。
  (b) リターン値が 4 バイト以下(cpu=300 時は 2 バイト以下)。

(6) (4)の関数呼び出しは、呼び出し元の関数での最終処理である。
(7) (5)の関数は、(3)の関数と同一ファイル内に定義されている。

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。
(1) goptimize オプションを指定してコンパイルする。
(2) 呼び出す関数の定義を別ファイルへ移動する。

6. 2 次元配列のアドレス演算結果不正

現象

2 次元配列のアドレス参照式に 2 回以上の加減算を実施した場合、正しい演算結果が生成されない場合があります。

```
[例]
  char *p,array[12][12];
  unsigned long x1,x2,x3;
  void sub()
  {
    p = (&array[x1][0] + x2 + x3);
     /* 配列のアドレス値ではなく、不正に配列データを参照します。 */
  }
```

発生条件

以下の条件をすべて満たす場合、発生することがあります。
(1) CPU 種別として 300, 300HN, 300HA, 2000N, 2000A, 2600N, 2600A のいずれかを指定している。
(2) 最適化なし(optimize=0)でコンパイルしている。
(3) 2 次元配列のアドレスに対して、2 回以上の加算、減算またはその組み合わせの演算を行っている。
(4) 2 次元配列のアドレスが、&配列[変数][0]の形式で記述されている。

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。
(1) 2 次元配列のアドレスを、配列[変数]の形式で求める。

```
[例]
p = (array[x1] + x2 + x3);
```

(2) 式を分割して、加算または減算を 1 回ずつ行う。

```
[例]
p = (&array[x1][0] + x2);
p = (p + x3);
```

(3) 最適化あり(optimize=1:デフォルト)でコンパイルする。

7. ビットフィールド設定・参照不正

現象

BFST/BFLD 命令を用いてビットフィールドへのアクセスを行った場合に、異なるメモリ位置をアクセスする場合があります。

[例]
<不正なコード>

```
    MOV.L #(_p+4),ER1 ; ここで ER1 に(_p+4)を設定
    BFLD #7,@ER1,R0L
    CMP.B #4:8,R0L
    BHI L29:8
    MOV.B #2:8,R0L
    BRA L38:8
    L31:
      :
    L35:
    BFLD #15,@ER3,R0L
    ADD.B #2:8,R0L
    MOV.L #(_p+2),ER3 ; ここで ER3 に(_p+2)を設定
    MOV.L ER3,ER2
    MOV.B R0L,R1L
    BFST R1L,#240,@ER2
    MOV.L ER3,ER1 ; ここで ER1 に ER3、すなわち(_p+2)を設定
    BFLD #240,@ER1,R2L
    MOV.B R2L,R0H
    BFST R0H,#15,@(_p+3):32
    L29:
    MOV.B #6:8,R0L
    L38:
    BFST R0L,#7,@ER1 ; ER1 は変更されることがあるので、
                       正しくは、BFST R0L,#7,@(_p+4)となります。
    SUB.B R0L,R0L
    SUB.B R0L,R0L
```

発生条件

以下の条件を全て満たす場合、発生することがあります。

(1) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。

(2) 最適化オプション(optimize=1:デフォルト)を指定している。

(3) ビットサイズが 7bit 以下でビットサイズ+ビットオフセットが 1byte 領域に収まるようなビットオフセットをもつビットフィールドを使用している。

(4) 同一ビットフィールドへの設定または参照が複数箇所で行われる。

(5) 同一ビットフィールドへの設定または参照が BFST/BFLD 命令を使用して行われる。

(6) 同一ビットフィールドへのアクセスに使用するアドレッシングモードの種別が異なる。

[例]

```
    BFST R0L,#7,@ER1
    BFST R0L,#7,@L3+3
```

回避策

該当箇所が存在した場合、以下の方法で回避してください。

・最適化なし(optimize=0)でコンパイルする。

8. ループ制御変数の置換不正

現象

2byte 以上の型のループ制御変数を持つループ中に 1byte の型の変数がある場合、オブジェクト不正となる場合があります。

[例]

```
  int a[100],b[100];
  int i;
  unsigned char x;
  void f(void){
    x = 3;
    i = 0;
    while (i <= 32760) { /* x をループカウンタとして使用し、
                            無限ループとなる。 */
      a[x] = b[x];
      x++;
      i+=4;
    }
  }
```

発生条件

以下の条件を全て満たす場合、発生することがあります。

(1) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。

(2) 最適化オプション(optimize=1:デフォルト)を指定している。

(3) 2byte 以上の型のループ制御変数(例での i)をもつループを記述している。

(4) ループ内に char 型または unsigned char 型の変数(例での x)がある。

(5) ループ制御変数に対して定数値の加減算を実施している。

(6) (4)の変数がインクリメントされている。

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。

(1) ループ内の変数の増分値を 1 でない値に変更する。

(2) 発生条件(4)の変数のサイズが、ループ制御変数以上のサイズとなるように型を変更する。

(3) ループ制御変数のサイズが、発生条件(4)の変数以下のサイズとなるように型を変更する。

9. 定数の不正伝播

現象

左辺、右辺が同一外部変数(非 volatile)の代入式がある場合、直前の if 文の then 節、あるいは else 節の値を不正に伝播する場合があります。

[例]

```
  int x;

  int sub(){
    if (x>=9999){
      x=1050;
    }
    x=x; /* 不要式として削除する */
    x++; /* 不正に 1050 を伝播して、x=1051 に変換 */

    return (x);
  }
```

発生条件

以下の条件を全て満たす場合、発生することがあります。

(1) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。

(2) 最適化オプション(optimize=1:デフォルト)を指定している。

(3) 左辺、右辺が同一外部変数(非 volatile)の代入式がある。

(4) (3)の代入式の前に if 文があり、then 節あるいは else 節のどちらか一方に当該変数への代入がある。

回避策

該当箇所が存在した場合、以下の方法で回避してください。

・opt_range=noblock オプションを指定する。

10. 文字列データの不正統合

現象

先頭メンバが文字列である構造体配列または多次元配列を初期値ありで複数定義し、その初期値の先頭に同一文字列を指定した場合、不正に文字列データを統合することがあります。

[例]

```
  /* 初期値の先頭が同一文字列"Hello"でかつサイズが共に 12byte */
  const char a[2][6] = {"Hello",{1,2,3,4,5,6} };
  const char b[2][6] = {"Hello",{6,5,4,3,2,1} };

  .SECTION C,DATA,ALIGN=2
    _a: ; static: a
    _b: ; static: b 不正に a に統合
  .SDATAZ "Hello"
```

```
    .DATA.B H'01,H'02,H'03,H'04,H'05,H'06

  /* 初期値の先頭が同一文字列"Hello"でかつサイズが共に 10byte */
  typedef struct {
    char a[6];
    long l;
  } st1;
  typedef struct {
    char a[6];
    char b[4];
  } st2;

  const st1 s1[] = {"Hello",1};
  const st2 s2[] = {"Hello", {0,1,2,3}};

  .SECTION C,DATA,ALIGN=2
    _ss: ; static: ss
    _tt: ; static: tt 不正に ss に統合
  .SDATAZ "Hello"
  .DATA.L H'00000001
```

発生条件

以下の条件を全て満たす場合、発生することがあります。

(1) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。

(2) 先頭メンバが文字列の構造体配列または多次元配列を const 指定ありで複数定義している。

(3) (2)の初期値の先頭が同一文字列である。

(4) (2)の構造体配列または多次元配列のサイズが同じである。

回避策

該当箇所が存在した場合、以下の方法で回避してください。

・const 指定をはずす。

11. cpuexpand 指定が有効にならない

現象

cpuexpand オプション指定でコンパイルしても、拡張仕様のコードが生成されない場合があります。

[例]

```
  -cpu=2600a -cpuexpand でコンパイル
  unsigned long ul1;
  unsigned int ui1;
  void sub()
  {
    ul1 = ui1 * -1 ;
```

<不正なコード>

```
_func:
  MOV.W @_ui1:32,R0
```

<正しいコード>

```
_func:
  MOV.W @_ui1:32,R0
```

```
NEG.W R0 ; 2byte*2byte(-1)→2byte      MOV.W #-1,E0
EXTU.L ER0 ; 4byte へゼロ拡張          MULXU.W E0,ER0 ; 2byte＊2byte→4byte
MOV.L ER0,@_ul1:32                    MOV.L ER0,@_ul1:32
```

発生条件

以下の条件を全て満たす場合、発生することがあります。

(1) CPU 種別として 300, 300HN, 300HA, 2000N, 2000A, 2600N, 2600A のいずれかを指定している。

(2) CPUEXPAND オプションを指定している。

(3) 下記の式を記述している。

(a) long = int(short) * (-1)

(b) unsigned long = unsigned int(unsigned short) * (-1)

回避策

該当箇所が存在した場合、以下のいずれかの方法で回避してください。

(1) 下記のようにキャストを挿入し、unsigned 型の場合は定数値を変更する。

[例]

```
long l1;
short s1;
unsigned long ul1;
unsigned short us1;
void sub(void){
  l1 = (long)s1 * (-1); // signed の場合、-1 の乗算を行う。
  ul1 = (unsigned long)us1 * 0xFFFF; // unsigned の場合、0xFFFF で乗算を行う。
}
```

(2) -1 を volatile 型の変数に代入し、その変数で乗算を実施する。

[例]

```
long l1;
short s1;
unsigned long ul1;
unsigned short us1;
void sub(void){
  volatile short x = (-1);

  l1 = s1 * x;
  ul1 = us1 * x;
}
```

(3) cpuexpand オプションを指定しない。

12. 4byte 以下の構造体転送不正

現象

4byte 以下の構造体において、その構造体のメンバが構造体配列のとき、構造体の転送命令が出力されない場合があります。 また 3byte の構造体配列のときは、内部エラーが出力される場合もあります。

[例]

```
  typedef struct {
    char c1;
    char d1;
  }ST1;

  typedef struct {
    char d1;
    ST1 sx[1];
  }ST3;

  ST1 G_S;

  ST3 sub()
  {
    ST3 st3;
    st3.sx[0] = G_S;
    return st3;
  }
```

〈不正なコード〉

```
_sub:
    SUBS #4,SP
    ; このコードが出力されない
    MOV.L @(8:2,SP),ER0
    MOV.W @SP,@ER0
    MOV.B @(2:2,SP),@(2:2,ER0)
    ADDS #4,SP
    RTS
```

〈正しいコード〉

```
_sub:
    PUSH.L ER6
    MOV.W @_G_S:32,R0
    MOV.L @(8:16,SP),ER1
    MOV.W R0,@ER1
    MOV.B R6H,@(2:16,ER1)
    POP.L ER6
    RTS
```

発生条件

以下の条件を全て満たす場合、発生することがあります。

(1) CPU 種別 H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。

(2) 最適化オプション(optimize=1:デフォルト)を指定している。

(3) 4byte 以下の局所変数を宣言している。

(4) (3)の構造体メンバに構造体配列がある。

(5) (4)の構造体配列に値を代入している。

回避策

該当箇所が存在した場合、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(optimize=0)でコンパイルする。

(2) 当該ローカル変数に volatile 修飾子を指定する。

株式会社 日立超LSIシステムズ