H8 C/C++コンパイラ Ver.6 使用上のご注意(2)

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.6.0.02 以降（コンパイラパッケージ V.6.00 Release 02 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

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, 6.0.01	6.0.00, 6.0.01
PSO08CAS6-SLR	6.0.00, 6.0.01	6.0.00, 6.0.01
PSO08CAS6-H7R	6.0.00, 6.0.01	6.0.00, 6.0.01

1. 引数の構造体/共用体メンバのアクセス不正

現象

structreg オプションを指定し、４バイト以下の引数の構造体/共用体メンバにアクセスした場合、正しく値が反映されない場合があります。

```
[例]
typedef struct{
      char stc_1;
      char stc_2;
      int  stc_3;
  }ST;

  void f045(ST p1_str, ST p2_str ) {
      ST *lp1;
      ST *lp2;

      lp1 = &p2_str;
      lp2 = &p2_str;
      p2_str.stc_2 = 2;          /* 不正コード出力部分 */

      sub(lp2);
        }

[不正コード]                              [正常コード]
  MOV.L   ER1,@(4:2,SP)
  MOV.L   ER0,@SP
  MOV.L   SP,ER0
  ADDS.L  #4,ER0
  MOV.B   #2:8,@(1:2,SP)        →    MOV.B  #2:8,@(5:2,SP)
```

発生条件

下記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) 4 バイト以下の構造体または共用体を引数として使用している。
(4) 引数がアドレス参照されている場合、もしくは関数内部でアドレス参照される場合。

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。
・引数をローカル変数に一旦代入し、その変数でアクセスする。

対象バージョン

Ver.6.0.00 以降

2. ポインタ比較式不正

現象

ポインタにキャストした定数同士を比較した場合、比較結果が異なる場合があります。

```
[例]
 int a;
  void f(void) {
    unsigned long t = 0xffffffff;
                        /* 0xffffffff が伝播され、
                        ((float *)0 < (float *)0xffffffff);となる。*/
    a = ((float *)0 < (float *)t);   /* 不正に a=0 となる     */
        }
```

発生条件

下記の条件を満たす場合、発生することがあります。
(1) 比較式がある。
(2) (1)の両辺が共にポインタ型にキャストされた定数である。
(3) (2)の定数のうち、少なくとも一方が signed long の範囲外(2147483648～4294967295)の値である。

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。
・一方の定数を volatile 付きの一時変数に格納し、その一時変数を用いて比較を行う。

```
  [例]
  int a;
  void f(void) {
    volatile unsigned long t = 0xffffffff;
    a = ((float *)0 < (float *)t);
  }
```

対象バージョン

Ver.6.0.00 以降

3. ビットフィールドの設定・参照不正

現象

ビットフィールドへの設定および参照を行った場合、値の設定や参照が正しく行えない場合があります。

```
[例]
typedef struct {
      char c:8;
      char c2:8;
      int i;
 }ST;
main()
{
      ST a;
      a.c=10;
      if (a.c==10) {     /* a.c を不正参照し、比較が FALSE になる    */
          func1();
           } else {
          func2();
      }
}
```

発生条件

以下の(1)、(2)いずれかの条件を全て満たす場合、発生することがあります。

(1)

- (a) CPU 種別として 300HN, 300HA のいずれかを指定している。
- (b) 最適化あり(optimize=1:デフォルト)でコンパイルしている。
- (c) 引数または局所変数の構造体を宣言している。
- (d) (c)の構造体が[unsigned] char 型で、サイズが 8bit のビットフィールドメンバを持つ。
- (e) (c)の構造体はレジスタ上に割りつき、(d)のビットフィールドメンバは En 上に割りついている。

(2)

- (a) CPU 種別として 300HN, 300HA, 2000N, 2000A, 2600N, 2600A のいずれかを指定している。
- (b) 最適化あり(optimize=1:デフォルト)でコンパイルしている。
- (c) 引数または局所変数の構造体を宣言している。
- (d) (c)の構造体が[unsigned] short または[unsigned] int で、サイズが 16bit のビットフィールドメンバを持つ。
- (e) (c)の構造体は、境界調整数が 1 である(pack=1 オプション、#pragma pack 1 が指定されている)。
- (f) (c)の構造体はレジスタ上に割りつき、(d)のビットフィールドメンバは En の下位 8bit と RnH に割りついている。

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) ビットフィールドの指定をやめ、スカラ型として宣言する。

[例]

```
struct ST {                          struct ST {
    char c:8;                            char c;
    char c2:8            →               char c2;
    int i;                               int i;
};                                   };
```

(2) 最適化なしでコンパイルする。

対象バージョン

Ver.4.0 以降

4. &構造体.配列[0] 等構造体メンバのアドレス参照時エラー

現象

&構造体.配列[0](&構造体->配列[0])形式で先頭アドレスを参照した場合、正しいアドレスが求められない場合があります。または内部エラーが出力される場合があります。

```
[例]
typedef struct ST {
      char array[12];
  }ST;

  ST st;
  int b,c;
  char *p;

void sub()
{
  p= &st.array[0] + b + c;
        /* アドレスではなく、st.array[0]の値を加算するコードを出力  */
}
```

発生条件

以下の(1)、(2)いずれかの条件を全て満たす場合、発生することがあります。

(1)

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

(b) 構造体メンバが配列で定義されている。

(c) (b)の先頭のアドレス値を用いて 2 回以上の加減算を実施する。

(d) 先頭アドレスは&構造体.配列[0]、または&構造体->配列[0]形式で求めている。

(2)

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

(b) 最適化あり(optimize=1:デフォルト)でコンパイルする。
(c) 変数の配列を定義している。
(d) (c)の先頭のアドレス値を用いて、2 回以上の加減算を実施する。
(e) 先頭アドレスを&配列[0]形式で求めている。

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。
・配列のアドレスを、構造体.配列(または構造体->配列)形式または、配列で求める。

対象バージョン

Ver.4.0 以降

5. &=0、|=0xFFFF の不正アドレスアクセス

現象

[unsigned]short/int 型の変数に対して複合論理演算を実施した場合、不正なアドレス(本来のアドレス+2)に値を設定するコードが生成される場合があります。

```
[例]
typedef struct {
    short      w1;
}*PST;
volatile PST pst = (PST)0xC40000;
short * volatile p;
void sub(void)
{
     pst->w1 &= 0;
    *p &= 0;
     pst->w1 |= 0xffff;
    *p |= 0xffff;
}
```

```
 [不正コード]                                    [正常コード]
_sub:
       MOV.L       @_pst:32,ER0
       SUB.W       R1,R1
       MOV.W       R1,@(2:16,ER0)     →    MOV.W  R1,@ER0
       MOV.L       @_p:32,ER0
       MOV.W       R1,@(2:16,ER0)     →    MOV.W  R1,@ER0
       MOV.L       @_pst:32,ER0
       MOV.W       #-1,R1
       MOV.W       R1,@(2:16,ER0)     →    MOV.W  R1,@ER0
       MOV.L       @_p:32,ER0
```

発生条件

下記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) 変数を[unsigned] short/int 型で宣言している。

(3) 変数は volatile 宣言されたポインタである。

(4) 下記の複合代入演算を記述している。

- 変数 &=0;
- 変数 |= 0xffff;

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) ポインタの指す先の領域に volatile を付加する。

[例]

```
typedef volatile struct {       /* volatile を付加する。    */
    short  w1;
}*PST;

volatile PST pst = (PST)0xC40000;
volatile short * volatile p;    /* volatile を付加する。    */
```

(2) &=,|=演算を単純代入(=)に変更する。

[例]

```
pst->w1 = 0;

*p = 0;
pst->w1 = 0xffff;
*p = 0xffff;
```

対象バージョン

Ver.3.0 以降

株式会社 日立超LSIシステムズ