1. 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る。

```
[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(a),(b)いずれかの条件を全て満たす場合、発生することがある。

(a)

(1) CPU 種別として 300HN, 300HA のいずれかを指定している。
(2) 最適化あり(optimize=1:デフォルト)でコンパイルしている。
(3) 引数または局所変数の構造体を宣言している。
(4) (3)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。
(5) (3)の構造体はレジスタ上に割りつき、iv.のビットフィールドメンバは En 上に割りついている。

(b)

(1) CPU 種別として 300HN, 300HA, 2000N, 2000A, 2600N, 2600A のいずれかを指定している。
(2) 最適化あり(optimize=1:デフォルト)でコンパイルしている。
(3) 引数または局所変数の構造体を宣言している。
(4) (3)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。
(5) (3)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)。
(6) (3)の構造体はレジスタ上に割りつき、iv.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

以下のいずれかの方法で回避してください。

(1) 最適化なしでコンパイルする。

(2) 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。

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

影響システム

H8S,H8/300 シリーズ C/C++コンパイラ Ver.4.0 以降

2. &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る。または内部エラーが出力される場合がある。

[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(a),(b)いずれかの条件を全て満たす場合、発生することがある。

(a)

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

(2) 構造体メンバが配列で定義されている。

(3) (2)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。

(4) 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。

(b)

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

(2) 最適化あり(optimize=1:デフォルト)でコンパイルする。

(3) 変数の配列を定義している。

(4) (3)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。

(5) 先頭アドレスを&配列[0]形式で求めている。

回避策

下記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。

影響システム

H8S,H8/300 シリーズ C/C++コンパイラ Ver.4.0 以降

3. &=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る。

[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]

```
 _sub:
      MOV.L     @_pst:32,ER0
      SUB.W     R1,R1
      MOV.W     R1,@(2:16,ER0)
      MOV.L     @_p:32,ER0
      MOV.W     R1,@(2:16,ER0)
      MOV.L     @_pst:32,ER0
      MOV.W     #-1,R1
      MOV.W     R1,@(2:16,ER0)
      MOV.L     @_p:32,ER0
      MOV.W     R1,@(2:16,ER0)
      RTS
```

[正常コード]

```
→    MOV.W  R1,@ER0
→    MOV.W  R1,@ER0
→    MOV.W  R1,@ER0
→    MOV.W  R1,@ER0
```

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る。
(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。

(a) 変数 &=0;

(b) 変数 |= 0xffff;

回避策

以下のいずれかの方法で回避してください。

(1) ポインタの指す先の領域に volatile を付加する。

```
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。

```
pst->w1 = 0;

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

影響システム

H8S,H8/300 シリーズ C/C++コンパイラ Ver.3.0 以降

4. 0 クリアコード不正削除

現象

分岐ごとに 0 を設定するコード(SUB 命令)を記述した場合、0 を設定するコードを不正に削除してオブジェクト不正となることがある。

[例]

```
        sub.b     R0H, R0H ; R0H に 0 を設定
              :
              :
      L55:
              :
        bls       L36
        sub.b     R0H, R0H ; R0H が 0 のままなので削除可能な命令となる。
        add.w     R0, R0   ; ここで R0H の値が変わる可能性がある。
              :
      L48:
        sub.b     R0H, R0H ; 不正に削除される
        add.w     R0, R0   ; R0H の値が 0 でない値の場合、オブジェクト不正となる。
```

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る。

(1) 最適化あり(optimize=1:デフォルト)を指定している。
(2) 分岐毎に 0 クリアを行うコードを記述している。

回避策

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。

影響システム

H8S,H8/300 シリーズ C/C++コンパイラ Ver.4.0.04 以降

株式会社 日立超LSIシステムズ