H8S,H8/300 シリーズ C/C++コンパイラ Ver.6.0.01 で対策した制限事項

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る。

```
[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る */
}
```

```
[不正なコード]              [正しいコード]
_sub:                       _sub:
  MOV.W   @_b+2:32,R0         MOV.L   @_b:32,ER0
  SHLR.W  #11:5,R0            SHLR.L  #11:5,ER0
  MOV.W   R0,@_a:32           MOV.W   R0,@_a:32
  RTS                         RTS
```

発生条件

次の(a),(b)いずれかを満たす場合、発生することがある。

(a) 次の条件をすべて満たす場合。

(1) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX 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) (unsigned char)(変数 / 定数) の式を記述している。

(4) 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。

(b) 次の条件をすべて満たす場合。

(1) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX 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) (unsigned int)(変数 / 定数) または(unsigned short)(変数 / 定数) の式を記述している。

(4) 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。

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る。

```
[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ない */
    };
```

発生条件

次の条件をすべて満たす場合、発生することがある。

(a) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
(b) 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。
(c) (b)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。
(d) (b)のポインタ型メンバの初期値がアドレス+定数である。
(e) (b)のスカラ型メンバの初期値が定数値である。
(f) 構造体が const 指定されている。

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る。

```
[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る */
}
```

```
[不正なコード]                    [正しいコード]
_func:                            _func:
  MOV.L  @_st:32,ER0                MOV.L  @_st:32,ER0
  AND.L  #h'dfffb3df:32,ER0         AND.L  #h'e00001ff:32,ER0
  OR.L   #h'0001fe00:32,ER0         OR.L   #h'0001fe00:32,ER0
  MOV.L  ER0,@_st:32                MOV.L  ER0,@_st:32
  RTS                               RTS
```

発生条件

次の条件をすべて満たす場合、発生することがある。

(a) CPU 種別として H8SXN,H8SXM,H8SXA,H8SXX のいずれかを指定している。
(b) ビットフィールドに定数値を代入している。
(c) 代入するビットフィールドが以下のいずれかである。
- 型:long ビット幅:20 ビットオフセット:3
- 型:long ビット幅:15 ビットオフセット:5

・型:long ビット幅:9 ビットオフセット:17
・型:long ビット幅:9 ビットオフセット:8
・型:long ビット幅:9 ビットオフセット:0
・型:int/short ビット幅:9 ビットオフセット:2

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る。

[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る */
}
```

発生条件

次の条件をすべて満たす場合、発生することがある。

(a) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
(b) structreg オプションを指定している。
(c) 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。

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る。

[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る       */
        } 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

次の条件をすべて満たす場合、発生することがある。

(a) 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。
(b) goptimize オプションを指定していない。
(c) 呼び出し元関数入口/出口にレジスタの退避/回復がない。
(d) (c)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。
(e) (d)の呼び出される関数は次を満たしている。
　・引数渡しにスタックを使用しない。
　・リターン値が 4 バイト以下(cpu=300 時は 2 バイト以下)。
(f) (d)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。
(g) (e)の関数は、(c)の関数と同一ファイル内に定義されている。

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る。

[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する */
}
```

発生条件

次の条件をすべて満たす場合、発生することがある。

(a) 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。
(b) 最適化なし(optimize=0)でコンパイルしている。
(c) 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。
(d) 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。

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る。

```
[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)
   SUB.B    R0L,R0L
   SUB.B    R0L,R0L
```

発生条件

次の条件をすべて満たす場合、発生することがある。

(a) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
(b) 最適化オプション(optimize=1:デフォルト)を指定している。
(c) 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。

(d) 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。
(e) 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。
(f) 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 ROL,#7,@ER1
      BFST ROL,#7,@L3+3
```

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る。

[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

次の条件をすべて満たす場合、発生することがある。

(a) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
(b) 最適化オプション(optimize=1:デフォルト)を指定している。
(c) 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。
(d) ループ内に char 型または unsigned char 型の変数(例での x)がある。
(e) 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。
(f) (d)の変数がインクリメントされている。

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る。

[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

次の条件をすべて満たす場合、発生することがある。

(a) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
(b) 最適化オプション(optimize=1:デフォルト)を指定している。
(c) 左辺、右辺が同一外部変数(非 volatile)の代入式がある。
(d) (c)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。

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る。

```
[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

次の条件をすべて満たす場合、発生することがある。

(a) CPU 種別として H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
(b) 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。
(c) (b)の初期値の先頭が同一文字列である。
(d) (b)の構造体配列または多次元配列のサイズが同じである。

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る。

```
[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]                                [正しいコード]
_func:                                       _func:
 MOV.W   @_ui1:32,R0                          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
 .DATA.L     H'00000001
```

発生条件

次の条件をすべて満たす場合、発生することがある。

(a) 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。
(b) CPUEXPAND オプションを指定している。
(c) 下記の式を記述している。
- long = int(short) * (-1)
- unsigned long = unsigned int(unsigned short) * (-1)

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る。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る。

[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
  .DATA.L      H'00000001
```

[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

次の条件をすべて満たす場合、発生することがある。

(a) CPU 種別 H8SXN, H8SXM, H8SXA, H8SXX のいずれかを指定している。
(b) 最適化オプション(optimize=1:デフォルト)を指定している。
(c) 4byte 以下の局所変数を宣言している。
(d) (c)の構造体メンバに構造体配列がある。
(e) (d)の構造体配列に値を代入している。

13. 内部エラー

現象

次の条件で内部エラーが発生することがある。

(a) ループ内で不変の配列参照があり、そのインデックス値に int(short)型の範囲を超える定数を使用した場合(C4098 出力)
(b) 変数と定数の演算を含む式が配列のインデックスに記述され、その配列を判定式の条件に使用した場合(C4098 出力)
(c) 先頭メンバが配列・構造体型の初期値ありローカル共用体を定義した場合(C4774 出力)

(d) 連続したループで 1 つ目のループ内の最初の文と 2 つ目のループの初期化条件が共通だった場合
(C4098 出力)
(e) 500 個以上の変数を宣言し、abs8/abs16 指定変数と指定なし変数が混在した場合(C4712 出力)
(f) (1byte 変数-定数)の式において、オーバーフローが発生する場合(C4722 出力)

14. オーバーフローを伴う演算判定

Ver.6.0.00 で注意事項としていた、"オーバフローを伴う演算判定"を対策しました。

15. cpuexpand オプションの解釈変更

次の 10 パターンは、H8SX では拡張命令の対象外となりますが、H8SX 以外の CPU では拡張命令の対象となります。

(a) signed long = signed int << constant
(b) signed long = unsigned int << constant
(c) unsigned long = signed int << constant
(d) unsigned long = unsigned int << constant
(e) signed int = (signed int << constant) / signed int
(f) signed int = (unsigned int << constant) / signed int
(g) signed int = (unsigned int << constant) / unsigned int
(h) unsigned int = (signed int << constant) / signed int
(i) unsigned int = (unsigned int << constant) / signed int
(j) unsigned int = (unsigned int << constant) / unsigned int

株式会社 日立超LSIシステムズ