RENESAS Tool News

RENESAS TOOL NEWS on November 1, 2006: 061101/tn1

Notes on Using the C/C++ Compiler Packages V4 through V.6 for the H8SX, H8S, and H8 MCU Families

We inform you of twelve problems in using the C/C++ compiler packages V4 through V.6 for the H8SX, H8S, and H8 MCU families.

1. Product Types Concerned

C/C++ compiler package V.4:
- PS008CAS4-MWR (Windows edition)
- PS008CAS4-SLR (Solaris edition)
- PS008CAS4-H7R (HP-UX edition)

C/C++ compiler package V.5:
- PS008CAS5-MWR (Windows edition)

C/C++ compiler package V.6:
- R0C40008XSW06R (Windows edition)
- R0C40008XSS06R (Solaris edition)
- R0C40008XSH06R (HP-UX edition)

2. Descriptions

2.1 Problem with Include Functions movfpe and movtpe (H8C-0045)

Versions concerned:
V.6.00 Release 00 through V.6.00 Release 03 and
V.6.01 Release 00 through V.6.01 Release 02

Symptom:
The movfpe and movtpe include functions may not properly be executed.

Conditions:
This problem may occur if the following conditions are all satisfied:

(1) Any of the following CPU options is selected (for example, -cpu=2000n on the command line):
2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX and AE5
Note, however, that the following cases are excluded:
- Any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected in V.6.00.
- The Code generation of Ver.4.0 Optimization technology option is selected in V.6.01 (-legacy=v4 on the command line).

(2) A call to include function movfpe or movtpe is made in a function more than once.

(3) The Optimization option is selected (the default statement -optimzize=1 represented on the command line).

Example 1:

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

Generated codes 1:

```
movfpe.b    @_a:16,r0l   ; movfpe executed only once
mov.b       r0l,@_data:32
```

Example 2:

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

Generated codes 2:

```
movfpe.b    @_a:16,r0l
```

```
 mov.b       r0l,@_data:32
 ; movtpe not executed
-------------------------------------------------
```

Workarounds:

Avoid this problem in any of the following four ways:

(1) Volatile-qualify the variable to be passed as an argument to movfpe and movtpe.

Example:

```
-------------------------------------------------
 #include <machine.h>
 #pragma abs16 a
 volatile char a, data; // Qualified to be volatile
 void main(void){
    movfpe(&a,data);
    movfpe(&a,data);
 }
-------------------------------------------------
```

(2) Do not select the Optimization option (-optimzize=0 on the command line).

(3) Select the External Variable Optimization option (-volatile on the command line).

(4) Select the Code generation of Ver.4.0 Optimization technology option (-legacy=v4 on the command line) if any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected in V.6.01.

2.2 Problem with Operations Where Type Conversions Are Made (H8C-0046)

Versions concerned:

V.6.01 Release 00 through V.6.01 Release 02

Symptom:

If a division or remainder operation is performed where conversions to type long are made, and then the type of the result of operation is again converted to the one before the first conversions are made, the result of operation will overflow.

Conditions:

This problem occurs if the following conditions are all satisfied:

(1) Any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected (for example, -cpu=2000n on the command line).

(2) The Code generation of Ver.4.0 Optimization technology option

(-legacy=v4 on the command line) is not selected.
(3) A division or remainder operation exists in the C source program.
(4) The types of the dividend and the divisor in (3) are both signed int or signed short and are converted to signed long.
(5) The result of the operation in (3) is assigned to a variable of type signed int or signed short.
(6) The values of the dividend and divisor in (3) are -32768 and -1 respectively.

Example:

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

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

Generated codes:

```
_main:
   ...
   mov.w      @_b:32,r0
   exts.l     er0
   mov.w      @_c:32,r1
   divxs.w    r1,er0        ; Overflown
   mov.w      r0,@_a:32
```

Workarounds:
Avoid this problem in either of the following ways:
(1) Select the Code generation of Ver.4.0 Optimization technology option (-legacy=v4 on the command line).
(2) Assign either the dividend or the divisor of a division or remainder operation to a volatile-qualified variable of type

signed long; then perform the operation.

Example:

```
---------------------------------------------------------------------
#include <stdio.h>
short a,b,c;
void main(void){
   volatile long lb; // Volatile-qualified
   init();
   lb = (long)b;
   a  = (lb / (long)c);
}

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

2.3 Problem with Inline Expansion of Library Functions strcpy and memcpy (H8C-0047)

Versions concerned:

V.6.00 Release 00 through V.6.00 Release 03 and
V.6.01 Release 00 through V.6.01 Release 02

Symptom:

If the Specific Library Function Inline Expansion option is used (-library=intrinsic on the command line) as well as library function strcpy or memcpy, the program is incorrectly executed.

Conditions:

This problem occurs if the following conditions are all satisfied:

(1) Any of the CPU options H8SXN, H8SXM, H8SXA, H8SXX, and AE5 is selected (for example, -cpu=h8sxn on the command line).
(2) The Specific Library Function Inline Expansion option for a specific library function is selected (-library=intrinsic on the command line).
(3) Library function strcpy or memcpy is used.
(4) Any of the following conditions is met:
   (a) Any of the registers except E4 is selected to be in scope of the #pragma global_register directive.

(b) When memcpy used, a value equal to or greater than 0x60001, or any constant address is passed as the third argument to this function.

Example 1:

```
-------------------------------------------------
 #include <string.h>
 char *dst;
 void test(void){
    strcpy(dst, (char *)0); // Conditions (3) and (4)-(b)
 }
-------------------------------------------------
```

Generated codes 1:

```
-------------------------------------------------
 _dst:      ; Code for strcpy not generated
 .RES.L     1
-------------------------------------------------
```

Example 2:

```
-------------------------------------------------
 #include <string.h>
 char *dst;
 void test(void){
    strcpy(dst, (char *)10); // Conditions (3) and (4)-(b)
 }
-------------------------------------------------
```

Generated codes 2:

```
-------------------------------------------------
 mov.w      #h'000a:16,r4; 10 bytes in size transferred
 mov.l      @_dst:32,er6
 mov.l      #h'000a:16,er5
 movsd.b    ($+4)
 rts/l      (er4-er6)
-------------------------------------------------
```

Workaround:

To avoid this problem, do not select the Specific Library Function Inline Expansion option (-library=intrinsic on the command line) for any specific library functions.

2.4 Problem with the #pragma inline_asm Directive (H8C-0048)

Versions concerned:

V.6.00 Release 00 through V.6.00 Release 03 and
V.6.01 Release 00 through V.6.01 Release 02

Symptom:

If any of the caller-save registers *1 is used within a function that has been declared to be #pragma inline_asm, the result of execution may become incorrect.

*1. The registers R0L, R0H, R0, E0, ER0, R1L, R1H, R1, E1, and ER1; for details of these, see Section 9.3.2 (3) "Rules concerning registers" in the User's Manual.

Conditions:

This problem may occur if the following conditions are all satisfied:
(1) Any of the following CPU options is selected (for example, -cpu=2000n on the command line):
2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX and AE5
Note, however, that the following cases are excluded:
- Any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected in V.6.00.
- The Code generation of Ver.4.0 Optimization technology option is selected in V.6.01 (-legacy=v4 on the command line).

(2) Among the Object Type options, the option called Assembly Source Code is selected (-code=asmcode on the command line).
(3) A function is declared to be #pragma inline_asm.
(4) Within the function in (3) is used an instruction for setting or updating the value of a caller-save register.
(5) In the program exists an expression that makes a call to the function in (3).
(6) The function in (3) does not have its return value, or the expression in (5), which makes a call to the function, does not refer to the return value of the function.
(7) The condition in (7)-1 or all the conditions in (7)-2 below are met.
(7)-1
Within the function containing the expression in (5) exist no expressions that make calls to any functions except the function in (3).
(7)-2
(a) In the program exists a function that is declared to be #pragma interrupt or __interrupt.
(b) The function in (a) is not declared to be #pragma regsave or __regsave

(c) All the functions to which calls are made in the function in (a) are expanded inline.

Example:

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

Generated codes:

```
-------------------------------------------------
 _f:
   mov.l      er0,er1 ; Values of arguments a and b copied to er1
   mov.l      #0,er1  ; Value of er1 overwritten
   add.w      e1,r1
   mov.w      r1,r0
   rts
-------------------------------------------------
```

Workarounds:

Avoid this problem in any of the following four ways:

(1) Use __asm and #pragma inline.

Example:

```
-------------------------------------------------
 #pragma inline (g)
 static void g(void) {
    __asm{
      MOV.L  #0,ER1
    }
 }
 int f(int a,int b) {
    g();
    return a+b;
 }
-------------------------------------------------
```

(2) Describe assembly codes using __asm.

Example:

```
-------------------------------------------------
 int f(int a,int b) {
    __asm{
       MOV.L  #0,ER1     // Describe codes for function
                         declared to be inline_asm
    }
    return a+b;
 }
-------------------------------------------------
```

(3) Declare the function that has already been declared to be #pragma interrupt again to be #pragma regsave or __regsave if Conditions (7)-2 are all met.

Example:

```
-------------------------------------------------
 #pragma regsave(func)
 #pragma interrupt(func)
-------------------------------------------------
```

(4) Define a dummy function that is not expanded inline; then make a call to it from a function declared to be #pragma interrupt if Conditions (7)-2 are all met.

Example:

```
-------------------------------------------------
 void dummy();
 void func(void){
    sub();         // Call to function declared to be inline_asm
    dummy();       // Call to dummy function
 }
 void dummy(){}
-------------------------------------------------
```

2.5 Problem with Constant Expressions Used as Subscripts (H8C-0049)

Versions concerned:

V.6.00 Release 00 through V.6.00 Release 03 and
V.6.01 Release 00 through V.6.01 Release 02

Symptom:

If an expression the result of evaluation of which is a constant expression is used as the subscript to an array, codes accessing incorrect area may be generated.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) Any of the CPU Options H8SXN, H8SXM, H8SXA, H8SXX, and AE5 is selected (for example, -cpu=h8sxn on the command line).

(2) An expression is used as the subscript to an array.

(3) The result of evaluation of the expression in (2) is a constant expression.

Example:

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

void main(){
   a[(x*0)+3].aaa = 20;// Conditions (2) and (3)
}
-------------------------------------------------
```

Generated codes:

```
-------------------------------------------------
_main:
  mov.w      #5:3,r0  ; Constant 5 is loaded in r0 instead of 3
  mova/l.l   @((_a+2):32,r0.w),er1
  mov.w      #h'0014:16,r0
  mov.w      r0,@er1
-------------------------------------------------
```

Workarounds:

Avoid this problem in either of the following ways:

(1) Assign the expression used as the subscript to an array to a volatile-qualified variable; then access the array.

Example:

```
-------------------------------------------------
struct {
   int pad;
   int aaa;
```

```
} a[20];
int x;
volatile unsigned long ul;// Volatile-qualified variable defined

void main(){
   ul = (x*0)+3;// Assigned to volatile-qualified variable
   a[ul].aaa = 20;
}
```

---

(2) Use a constant as the subscript to an array.

Example:

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

void main(){
   a[3].aaa = 20;// Constant used as subscript
}
```

2.6 Problem with Assignments of 3-Byte Variables or Members of a Structure or Union (H8C-0050)

Versions concerned:

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

Symptom:

If a 3-byte variable or member of a structure or union that is located in memory or the stack is copied to a register, a 1-byte variable located in the register takes an incorrect value.

Conditions:

This problem may occur if all the conditions in (1)-1 or those in (1)-2 below are satisfied.
(1)-1

(a) The version of the compiler is V.6.01 Release 00, 01, or 02.
(b) Any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected (for example, -cpu=2000n on the command line).
(c) The Code generation of Ver.4.0 Optimization technology option (-legacy=v4 on the command line) is not selected.
(d) The Optimization for speed options (-speed=expression and -speed=struct on the command line) are not selected.
(e) A 3-byte variable or member of a structure or union is located in memory or the stack.
(f) A 3-byte variable or member of a structure or union is located in a register.
(g) In a 1 byte of the register where the variable or member in (f) is located, another variable is also located.
(h) An expression exists which assigns the variable or member in (e) to the one in (f).
(i) After the assignment expression in (h) is performed, an assignment is made to the variable or member in (f).

(1)-2
(a) Any of the following CPU options is selected (for example, -cpu=300hn on the command line):
300HN, 300HA, 2000N, 2000A, 2600N, and 2600A
(b) The Code generation of Ver.4.0 Optimization technology option is selected in V.6.01 (-legacy=v4 on the command line).
(c) A structure or union variable is used as a local variable or an argument.
(d) The variable in (c) is 4 bytes wide and its first member is a structure or union type of 3 bytes wide.
(e) The variable in (c) is located in a register.

Example:

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

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

ST3  m_st = { 0, 1, 2 };                // Condition (1)-1 (e)
```

```
ST4 sub(ST4  r_st){
   r_st.st3 = m_st;                    // Condition (1)-1 (h)
   return(r_st);
}
void main(void){
   ST4  r_st = { { -1, -1, -1 }, -1 };     // Conditions (1)-1 (f)
                                           and (g)

   r_st = sub(r_st);

   if(r_st.c3==-1){                    // Condition (1)-1 (i)
    ...
   } else {
    ...
   }
}
```

Generated codes:

```
_sub:
   push.l  er2
   mov.l   #_m_st:32,er0
   jsr     @$mv3mr$:24        ; 3 bytes copied
   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          ; Value in R2L overwritten
```

Workarounds:

(1) Avoid this problem in either of the following ways if Conditions (1)-1 are all met:

(a) Select the Optimization for speed option (-speed=expression or

-speed=struct on the command line).

(b) Declare the function containing the assignment expression in Condition (1)-1 (h) to be #pragma option speed=struct or #pragma option speed=expression.

Example:

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

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

(2) Avoid this problem in either of the following ways if Conditions (1)-2 are all met:

(a) Add a volatile qualifier to the local variable or argument in Condition (1)-2 (c).

(b) Change the order of declaring structure or union member variables.

Example:

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

(c) Add a dummy member variable of 1 byte wide or more to the structure or union in order to make the size of the variable in Condition (1)-2 (c) grater than 4 bytes.

Example:

```
---------------------------------------------------
typedef struct {
   ST3  st3;
   char c3;
   char dummy;      /* Add dummy member variable of 1 byte
```

```
                    or more to structure */
  }ST4;
  ------------------------------------------------
```

2.7 Problem with Use of the #pragma option speed=register Directive (H8C-0051)

Versions concerned:

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

Symptom:

If the register option is used as a sub option of #pragma option speed, an incorrect stack area may be accessed.

Conditions:

This problem occurs if the following conditions are all satisfied:
(1) Any of the CPU options 300, 300HN, and 300HA is selected (for example, -cpu=300 on the command line).
(2) The Optimization option is selected (the default statement -optimzize=1 represented on the command line).
(3) The #pragma option speed statement is used, or "register" is selected as a sub option of #pragma option speed (#pragma option speed=register).
(4) One or more guaranteed registers *1 are used, and the codes for saving or restoring these registers are generated.
(5) An access is made to an argument located in the stack.

*1. For details of the registers of this type, see Section 9.3.2 (3) "Rules concerning registers" in the User's Manual.

Example:

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

#pragma option speed=register            // Condition (3)
void function(union_data* temp,long long_data,char index){
```

```
    short int_a;
    short int_b;

    int_a = temp->int_a;              // ER6 used
    int_b = temp->int_b;

    data[index].int_a = int_a * long_data; // Condition (5)
    data[index].int_b = int_b * long_data* 16384;
}
```

Generated codes:

```
_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 ; Incorrect stack area accessed
    ...
```

Workarounds:

Avoid this problem in any of the following ways:

(1) Do not select the Optimization option (-optimzize=0 on the command line).

(2) Do not use #pragma option speed and #pragma option speed=register but select the Optimization for speed option only (-speed=expression and -speed=struct on the command line).

(3) Do not use #pragma option speed=register.
If you want to use #pragma option speed, specify it as follows:
#pragma option speed=shift,loop,switch,inline,struct,expression

2.8 Problem with Comparison between Two Bit Field Members of 1 Bit Wide (H8C-0052)

Versions concerned:

V.6.00 Release 00 through V.6.00 Release 03 and
V.6.01 Release 00 through V.6.01 Release 02

Symptom:

When two bit field members of 1 bit wide in a structure or union are compared, an incorrect result may be obtained.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) Any of the following CPU options is selected (for example, -cpu=2000n on the command line):
2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, and AE5

Note, however, that the following cases are excluded:
- Any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected in V.6.00.
- The Code generation of Ver.4.0 Optimization technology option is selected in V.6.01 (-legacy=v4 on the command line).

(2) A structure or union containing bit field members of 1 bit wide is declared and defined.

(3) In the program exists an expression that makes a comparison between two bit field members of 1 bit wide in the structure or union in (2).

(4) The comparison in (3) is made using either of the equality operators != and ==.

Example:

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

int x;
struct ST st1;
struct ST st2[50];
void temp(){
   if (st1.bit == st2[x].bit) { // Conditions (3) and (4)
      sub();
   }
}
---------------------------------------------------
```

Generated codes:

```
---------------------------------------------------
_temp:
   bld.b      #7,@_st1:32 ; Carry flag set
   mov.w      @_x:32,r0
   exts.l     er0
   shll.l     er0         ; Carry flag overwritten
   mov.b      @(_st2:32,er0),r1l
```

```
    bxor.b      #7,r1l
    bcs         L28:8
```

Workarounds:
Avoid this problem in either of the following ways:
(1) Assign the addresses of two variables of a structure or union to pointers; then access their members using the pointers.

Example:

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

struct ST *p1, *p2; // Pointer variables declared
int x;
struct ST st1;
struct ST st2[50];
void temp(){
    p1 = &st1;
    p2 = &st2[x];
    if( p1->bit == p2->bit) {
        sub();
    }
}
```
---

(2) Assign the members of two variables of a structure to two volatile-qualified variables of the same type; then compare them.

Example:

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

int x;
struct ST st1;
struct ST st2[50];
void temp(){
    volatile int a, b; // Volatile-qualified variables declared
```

```
    a = st1.bit;
    b = st2[x].bit;
    if( a == b) {
       sub();
    }
  }
```

2.9 Problem with Iteration Statements (H8C-0053)

Versions concerned:

V.6.00 Release 00 through V.6.00 Release 03 and
V.6.01 Release 00 through V.6.01 Release 02

Symptom:

If an expression that may cause any exceptions to arise exists in a program flow not executed in an iteration statement, this expression may be performed.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) Any of the following CPU options is selected (for example, -cpu=2000n on the command line):
2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX and AE5
Note, however, that the following cases are excluded:
- Any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected in V.6.00.
- The Code generation of Ver.4.0 Optimization technology option is selected in V.6.01 (-legacy=v4 on the command line).

(2) The Optimization option is selected (the default statement -optimzize=1 represented on the command line).

(3) In the program exists an iteration statement.

(4) The iteration statement in (3) contains a division or remainder expression.

(5) The division or remainder expression in (4) satisfies all the following **Conditions:**
(a) Its divisor is 0 or an expression the result of whose evaluation is 0.
(b) It is an expression containing no variable whose value is updated in the iteration statement in (4).
(c) It is an expression not executed in the iteration statement in (4).

Example:

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

Workarounds:

Avoid this problem in any of the following three ways:

(1) Qualify any variable in the division or remainder expression to be volatile.

Example:

```
-------------------------------------------------
 int a[100];
 int W=1, X=0, Z;
 volatile int Y; // Volatile-qualified
 void f()
 {
    int i;
    for (i=0;i<100;i++){
       if (X > 0) {
          Y = Z / (W-1);
       }
       a[i] =1 ;
    }
 }
-------------------------------------------------
```

(2) Remove the division or remainder expression.

Example:

```
-------------------------------------------------
int a[100];
int W=1; X=0, Z;
void f()
{
   int i;
   for (i=0;i<100;i++){
     // Expression not performed is removed
      a[i] =1 ;
   }
}
-------------------------------------------------
```

(3) Do not select the Optimization option (-optimzize=0 on the command line).

2.10 Problem with the Direct Assignment of a Return value of a Function to a Member of a Structure (H8C-0054)

Versions concerned:

V.6.00 Release 00 through V.6.00 Release 03 and
V.6.01 Release 00 through V.6.01 Release 02

Symptom:

If a return value of a function is assigned to a member of a structure straight forwards, the values held in other members declared in the same structure may be affected.

Conditions:

This problem occurs if the following conditions are all satisfied:

(1) Any of the following CPU options is selected (for example, -cpu=2000n on the command line):
2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX and AE5
Note, however, that the following cases are excluded:
- Any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected in V.6.00.
- The Code generation of Ver.4.0 Optimization technology option is selected in V.6.01 (-legacy=v4 on the command line).

(2) The Optimization option is selected (the default statement -optimzize=1 represented on the command line).

(3) A structure the sizes of whose variables are 4 bytes or less is declared and defined.

(4) The variables of the structure in (3) are declared to be
local ones.
(5) The variables of the structure in (3) or their members are
not qualified to be volatile.
(6) A return value of the function is assigned to a member of
the structure in (3).

Example:

```
#include <stdio.h>
typedef unsigned char   UC;
typedef unsigned short  US;
typedef unsigned long   UL;

typedef union _UDWORD       // Conditions (3) and (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;      // Conditions (4) and (5)
   UC   dmy1, dmy2;

   Size.BYTE.eh = setval(); // Condition (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;
    }
 }
```

Generated codes:

```
_func:                          ; function: func
    ...
    jsr       @er2
    mov.b     r0l,@(H'000B:16,sp)
    jsr       @er2
    mov.w     e1,r3         ; Whole of structure not loaded in er1
    mov.b     r0l,r3h
    mov.w     r3,e1
    mov.l     er1,@(12:2,sp); Whole of structure saved on stack
```

Workarounds:

Avoid this problem in any of the following ways:

(1) Do not select the Optimization option (-optimzize=0 on the command line).

(2) Select #pragma option nooptimize for the function concerned.

(3) Qualify the structure variables to be volatile.

(4) Make the sizes of the structure variables greater than 4 in bytes.

2.11 Problem with __asm (H8C-0055)

Versions concerned:

V.6.01 Release 00 through V.6.01 Release 02

Symptom:
Using __asm may generate non-existent instructions,
resulting in execution errors.

Conditions:
This problem occurs if the following conditions are all satisfied:
(1) Any of the CPU options 2000N, 2000A, 2600N, and 2600A is
selected (for example, -cpu=2000n)
(2) A jsr or jmp instruction exists in __asm.
(3) The destination of the instruction in (2) is represented in
a 32-bit absolute address.

Example:

```
---------------------------------------------------------------------
extern void sub1(void);
extern void sub2(void);
void func(void)
{
   __asm{
      jsr @sub1:32 // Conditions (2) and (3)
      jmp @sub2:32 // Conditions (2) and (3)
   }
}
---------------------------------------------------------------------
```

Workarounds:
Avoid this problem in either of the following ways:
(1) Select the option called Assembly source code among the Object
Type options (-code=asmcode on the command line).

(2) Represent the absolute addresses of the jsr and jmp instruction
in 24 bits wide.

Example:

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

```
}
```

2.12 Problem with the Initial Value of the Address of a Function (H8C-0056)

Versions concerned:

V.6.00 Release 00 through V.6.00 Release 03 and
V.6.01 Release 00 through V.6.01 Release 02

Symptom:

Consider the case where the address of a function is cast to a pointer having an arbitrary data type and then a constant is added to or subtracted from it. If the result of this operation is used as the initial value of a variable in a static data storage, an incorrect result will be obtained.

Conditions:

This problem occurs if the following conditions are all satisfied:

(1) Any of the following CPU options is selected (for example, -cpu=2000n on the command line):
2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX and AE5
Note, however, that the following cases are excluded:
- Any of the CPU options 2000N, 2000A, 2600N, and 2600A is selected in V.6.00.
- The Code generation of Ver.4.0 Optimization technology option is selected in V.6.01 (-legacy=v4 on the command line).

(2) The symbolic name of a function or the one preceded by an address operator is cast to a pointer having an arbitrary data type; then a constant is added to or subtracted from it.

(3) A variable in a static data storage is defined which takes the result of the operation in (2) as its initial value.

Example:

```
---------------------------------------------------------------------
void func();
char *p = (char*)func + 1;  // Conditions (2) and (3)
---------------------------------------------------------------------
Generated codes:
---------------------------------------------------------------------
_p:                             ; static: p
  .DATA.L     _func ; Constant H'00000001 not added
---------------------------------------------------------------------
```

Workaround:

To avoid this problem, declare the name of the function to be the name of an array of an arbitrary data type in another file; then select the option called Assembly source code among the Object Type options (-code=asmcode on the command line).

Example

```
---------------------------------------------------------------------
// Described in another file
extern int func[];
char *p = (char*)func + 1;
---------------------------------------------------------------------
```

3. Schedule of Fixing the Problems

We plan to fix the above problems in the next release of the product, the C/C++ compiler package V.6.01 Release 03.
