RENESAS Tool News

RENESAS TOOL NEWS on June 23, 2008: 080623/tn1

Notes on Using the C/C++ Compiler Packages for the H8SX, H8S, and H8 MCU Families V.4, V.5, and V.6

Please take note of the twelve problems described below in using the C/C++ compiler package for the H8SX, H8S, and H8 MCU families V.4, V.5, and V.6.

1. With Declaring a Structure, a Union, or Their Member Variables to Be Accessed on Keyword __evenaccess (H8C-077)
2. With Using Library Functions scanf(), fscanf(), and sscanf() (H8C-078)
3. With Using the Same Variable More Than Once in an Expression (H8C-079)
4. With Accessing Members of a Structure or Union with the Alignment Number of 1. as (H8C-080)
5. With Coding Multi-Loops with the Vacant Loop Elimination Option del_vacant_loop=1 Selected (H8C-081)
6. With Putting the Same statement in More Than One Block (H8C-082)
7. With Assigning the Return Value of a Function to a Member Variable of a Structure, Union, or Class (H8C-083)
8. With Destination Addresses of the jmp Instruction (H8A-0001)
9. With Placing Labels in an Absolute Section (H8A-0002)
10. With Calling an Assembler Routine with the Number of Arguments-Storing Registers Being Declared to Be 3 (LNK-0001)
11. With Referencing the Initial Value of a Variable with the Short Absolute Addressing Mode Option Selected (LNK-0002)
12. With Using the Same Code Unification Option (LNK-0003)

1. Product, Versions, and Product Type Concerned

- Product: C/C++ compiler package for the H8SX, H8S, and H8 MCU families
- Versions: V.4.0 through V.6.02 Release 00
- Product Types:

V.4:

PS008CAS4-MWR (Windows edition)
PS008CAS4-SLR (Solaris edition)
PS008CAS4-H7R (HP-UX edition)

V.5:

PS008CAS5-MWR (Windows edition)

V.6:

R0C40008XSW06R (Windows edition)

R0C40008XSS06R (Solaris edition)

R0C40008XSH06R (HP-UX edition)

2. Seven Problems in C/C++ Compiler

2.1 With Declaring a Structure, a Union, or Their Member Variables to Be Accessed on Keyword __evenaccess (H8C-077)

Versions Concerned:

V.6.01 Release 00 through V.6.02 Release 00

Symptom:

The __evenaccess declaration made for a structure, a union, or their member variables may become ineffective.

Conditions:

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

(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, AE5, or RS4 is selected (for example, -cpu=2000N in the command line).

(2) As a CPU option, 2000N, 2000A, 2600N, or 2600A is selected, and the Compatibility of output code option -legacy=v4 not selected.

(3) Any one of the conditions (A), (B), (C), and (D) below is satisfied.

(A) All the conditions from (A-1) to (A-5) below are met.

(A-1) A structure or union is declared; then another structure or union nested in a member of the above is declared.

(A-2) A member of the nested structure or union in (A-1) is of type int equal to or less than 4 bytes.

(A-3) The structure or union nested in (A-1) is declared to be of an array or pointer type.

(A-4) A structure- or union-type pointer is declared which points to the nesting structure or union declared in (A-1), and also the pointer is declared to be accessed on __evenaccess.
Or, the nesting structure or union or their members are declared to be accessed on __evenaccess.

(A-5) The area necessary to the nesting structure or union in (A-1) is not yet reserved.

(B) All the conditions from (B-1) to (B-4) below are met.

(B-1) A structure or union is declared, and 1 is used

as the alignment number for the structure or union (-pack=1 or #pragma pack 1).

(B-2) The members of the structure or union in (B-1) are 2 bytes or 4 bytes long except that they are of the structure or bit field type.

(B-3) A structure- or union-type pointer is declared which points to the structure or union declared in (B-1), and also the pointer is declared to be accessed on __evenaccess.
Or, the structure or union or their members are declared to be accessed on __evenaccess.

(B-4) The area necessary to the structure or union declared in (B-1) is not yet reserved.

(C) All the conditions from (C-1) to (C-3) below are met.

(C-1) A local variable or argument of a structure or union is defined, where the structure or union is 4 bytes or less in size.

(C-2) A stack area is allocated to the structure or union in (C-1).

(C-3) A member of the structure or union in (C-1) is declared to be accessed on __evenaccess.

(D) All the conditions from (D-1) to (D-3) below are met.

(D-1) A structure or union is declared a member of which is a bit field of 1 bit long.

(D-2) A structure- or union-type pointer is declared which points to the structure or union declared in (D-1), and also the pointer is declared to be accessed on __evenaccess.
Or, the structure or union or their members are declared to be accessed on __evenaccess.

(D-3) Either the condition (D-3-1) or (D-3-2) below is met.

(D-3-1) A variable of the structure or union in (D-1) a member of which are a bit field of type unsigned long is defined as a global or local type.
Or, a structure- or union-type pointer is declared which points to the above structure or union.

(D-3-2) The size of the structure or union in (D-1) a member of which is a bit field of type unsigned short, signed short, unsigned int, or signed int is 4 bytes or less.

(4) A member of the structure or union described in (3) is referenced.

Example 1.

```
-----------------------------------------------------------
struct S {
   struct SS{              // Condition (3)-(A-1)
      unsigned short c:1;   // Condition (3)-(A-2)
   }a[2];                  // Condition (3)-(A-3)
};
#define STR (*(struct S __evenaccess *)0x011000)
                                    // Condition (3)-(A-4)
void func()
{
   STR.a[0].c = 1;   // Condition (4)
}
```

Generated Code:

```
-----------------------------------------------------------
_func    MOV.L     #H'00011000:32,ER0
         BSET.B    #7:3,@ER0 ; Not accessed using declared length
                     ; (2 bytes).
         RTS
```

Example 2.

```
-----------------------------------------------------------
#pragma pack 1          // Condition (3)-(B-1)
struct B {
   unsigned char UC1;
   unsigned short US1;  // Condition (3)-(B-2)
};
#define OBJB (*(volatile struct B __evenaccess *)0xFFFF23)
                                  // Condition (3)-(B-3)
void main(){
   OBJB.US1 = 1;                      // Condition (4)
}
```

Generated Code:

```
-----------------------------------------------------------
_main    MOV.W     #H'0001,R1
         MOV.B     R1H,@H'00FFFF24:8  ; Not accessed using declared
         MOV.B     R1L,@H'00FFFF25:8  ; length (2 bytes).
          ;
```

Example 3.

```
-----------------------------------------------------------
struct _str {
```

```
    __evenaccess long a ;  // Condition (3)-(C-3)
} ;
void func()
{
    volatile struct _str lstr ;  // Conditions (3)-(C-1), (3)-(C-2)
    if ( lstr.a & 0x80000000 ) { // Condition (4)
        sub();
    }
}
```

Generated Code:

```
---------------------------------------------------------------
_func    SUBS.L    #4,ER7
         MOV.B     @ER7,R0L ; Not accessed using declared
                            ; length (2 bytes).
```

Example 4.

```
---------------------------------------------------------------
// -cpu=h8sxa
struct ST{                          // Condition (3)-(D-1)
  __evenaccess unsigned long DATA:1; // Condition (3)-(D-2)
}st;
int DATA_int;                       // Condition (3)-(D-3-1)
void func (void){
  DATA_int = st.DATA;
}
```

Generated Code:

```
---------------------------------------------------------------
_func  SUB.L ER1,ER1
       BLD   #7,@st:32 ; Not accessed using declared length (4 bytes).
       BST   #0,R1L
       MOV.W R1,@_DATA_int:32
```

Workarounds:

(a) If conditions (1), (2), (3)-(A), and (4) are all satisfied, avoid the symptom in either of the following ways:

(a-1) Declare the structure or union to be __evenaccess whose necessary area is reserved without using the absolute address representation.

Example:

```
---------------------------------------------------------------
```

```
struct S {
   struct SS{
      unsigned short c:1;
   }a[2];
};
#pragma address STR=0x011000 // #pragma address reserves STR area.
__evenaccess struct S STR;   // STR declared to be __evenaccess.
void func()
{
   STR.a[0].c = 1;
}
```

(a-2) Declare the structure or union to reference to be __evenaccess.

Example:

```
-----------------------------------------------------------------
struct S {
   struct SS{
      __evenaccess unsigned short c:1; // Structure to reference
                                       // declared to be
                                       // __evenaccess.
   }a[2];
};

#define STR (*(struct S *)0x011000) // __evenaccess canceled.
-----------------------------------------------------------------
```

(b) If conditions (1), (2), (3)-(B), and (4) are all satisfied,
avoid the symptom as follows:
Declare the structure or union member variable of 2 or 4 bytes long
to be a bit field except for a variable of type float; then use it.

Example:

```
-----------------------------------------------------------------
struct B {
   unsigned char UC1;
   unsigned short US1:16; // Declared to be a bit field.
};
```

(c) If conditions (1), (2), (3)-(C), and (4) are all satisfied,
avoid the symptom as follows:
Add a dummy member to the structure or union so that its size
can exceed 4 bytes.

Example:

```
------------------------------------------------------------
struct _str {
    __evenaccess long a;
    char dummy;         // A dummy member added to structure
                        // so that its size can exceed 4 bytes.
} ;
```

(d) If conditions (1), (2), (3)-(D), and (4) are all satisfied,
avoid the symptom as follows:
(d-1) If condition (D-3-1) met, make the bit field 2 bits long
or more.
(d-2) If condition (D-3-2) met, make the bit field 2 bits long
or more, or add a dummy member to the structure or union
so that its size can exceed 4 bytes.

Example:

```
------------------------------------------------------------
struct ST{
  __evenaccess unsigned long DATA:2;
}st;
```

2.2 With Using Library Functions scanf(), fscanf(), and sscanf() (H8C-078)

Versions Concerned:

V.4.0 through V.6.02 Release 00

Symptom:

If the 's' conversion specifier is used for the conversion performed by scanf(), fscanf(), and sscanf(), the blank space (' '), newline ('¥n') and horizontal tab ('¥t') characters will be converted and stored in the storage area pointed to by argument ptr because the above characters are interpreted as those to be converted.

Conditions:

This symptom occurs if the following conditions are all satisfied:
(1) Any of the library functions scanf(), fscanf(), and sscanf() is used.
(2) The 's' conversion specifier is selected.
(3) In the character string to be converted exists a blank space (' '), newline ('¥n'), or horizontal tab ('¥t') character.

Example:

```
------------------------------------------------------------
void main( void )
{
   char tmp[ 10 ];
   sscanf( "ABC DEF", "%9s", tmp );    // Conditions (1), (2)
   printf( "%s¥n", tmp );              // "ABC DEF" displayed.
}
```

Workarounds:

Avoid the symptom as follows:
Select the '[' conversion specifier instead of 's' so that
the blank space, newline, and horizontal tab characters can be
excluded from the characters to be converted.

Example:

```
------------------------------------------------------------
void main( void )
{
   char tmp[ 10 ];
   sscanf( "ABC DEF", "%9[^ ¥t¥n]", tmp );  // Specifier [ used
                                  // for s; then put in
                                  // blank space, newline,
                                  // and horizontal tab.
   printf( "%s¥n", tmp );                 // "ABC" displayed.
}
```

2.3 With Using the Same Variable More Than Once in an Expression (H8C-079)

Versions Concerned:

V.6.01 Release 00 through V.6.02 Release 00

Symptom:

If the same variable is used more than once in an expression,
incorrect results of operations may be produced.

Conditions:

This symptom may occur if the following conditions are all satisfied:
(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA,
H8SXX, AE5, or RS4 is selected (for example, -cpu=2000N in the
command line).
(2) As a CPU option, 2000N, 2000A, 2600N, or 2600A is selected, and
the Compatibility of output code option -legacy=v4 not selected.

(3) The -optimize=1 option is selected.
(4) The same variable is used more than once as operands in an expression performing an addition, subtraction, multiplication, division, logical AND, logical OR, or exclusive OR operation.

Example:

```
---------------------------------------------------------------
unsigned long func2(unsigned short a);
unsigned long func1(unsigned long m)
{
   unsigned long r=0,tl=0;
   unsigned short r0=0,r1=0,t=0;
   unsigned short m0=0,m1=0;
   r=func2(m>>16);
   r0=r>>16;
   m1=m;
   tl=r0*r0; // Condition (4), r0*r0 gives an incorrect result.
   t=tl>>16;
   tl=((unsigned long)m1*t+0x0000)>>15;
   r-=(unsigned long)tl<<0;
   return r-1;
}
```

Generated Code:

```
---------------------------------------------------------------
_func1:
        STM.L      (ER2-ER3),@-SP
        MOV.L      ER0,ER3
        MOV.W       E3,R0
        BSR        _func2:8
        MOV.L      ER0,ER2
        MULXU.W     E0,ER1   ; Multiplied with ER1 set to no value.
        MOV.W       E1,E0
        MOV.W       E0,R1
        EXTU.L     ER1
        MOV.W       R3,R0
        EXTU.L     ER0
        MULU.L      ER0,ER1
        SHLR.L     #15:5,ER1
        SUB.L      ER1,ER2
        DEC.L      #1,ER2
        MOV.L      ER2,ER0
        RTS/L      (ER2-ER3)
```

Workarounds:

Avoid the symptom in either of the following ways:

(1) Use the -optimize=0 option instead of -optimize=1.

(2) Place the preprocessor statement "#pragma option nooptimize" immediately before the function where the symptom arises.

2.4 With Accessing Members of a Structure or Union with the Alignment Number of 1. as (H8C-080)

Versions Concerned:

V.6.01 Release 00 through V.6.02 Release 00

Symptom:

No members of a structure or union for which 1 is selected as the alignment number can be accessed by 1 byte.

Conditions:

This symptom occurs if the following conditions are all satisfied:

(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, or RS4 is selected (for example, -cpu=2600A in the command line).

(2) The Compatibility of output code option -legacy=v4 is not selected.

(3) As the alignment number for a structure or union, 1 is used (-pack=1 or #pragma pack 1).

(4) Condition (A) or (B) below is satisfied:

(A) All the conditions from (A-1) to (A-3) below are met.

(A-1) The structure or union in (3) is 4 bytes in size.

(A-2) A 4-byte member is defined in the structure or union in (A-1).

(A-3) A variable of the structure or union in (A-1) is declared to be a local variable.

(B) All the conditions from (B-1) to (B-4) below are met.

(B-1) A structure- or union-type pointer is declared which points to the structure or union in (3).

(B-2) A 2- or 4-byte bit field of type int is defined as a member of the structure or union in (B-1).

(B-3) An odd address is allocated to a member of the structure or union in (B-1).

(B-4) The area for the structure or union in (B-1) is not yet reserved.

(5) A member in (4)-(A-2) or -(B-2) is referenced.

Example 1.

```
---------------------------------------------------------------
#pragma pack 1                // Condition (3)
struct _str {
   long a ;                   // Conditions (4)-(A-1), (4)-(A-2)
} ;
void func(long p)
{
   volatile struct _str lstr ; // Condition (4)(A-3)
   p = lstr.a ;               // Condition (5)
}
```

Generated Code:

```
---------------------------------------------------------------
  _func    SUBS.L    #4,ER7
           MOV.L     @ER7,ER0 ; Not accessed by 1 byte.
```

Example 2.

```
---------------------------------------------------------------
#pragma pack 1 // Condition (3)
struct B {
   unsigned short US1:16; // Condition (4)-(B-2)
};
#define OBJB (*(volatile struct B*)0xFFFF23)
                     // Conditions (4)-(B-1), (4)-(B-3)
void main(){
   OBJB.US1 = 1;         // Condition (5)
}
```

Generated Code:

```
---------------------------------------------------------------
_main  MOV.W  #H'0001,R1
       MOV.W  R1,@H'00FFFF23:16  ; Not accessed by 1 byte.
       RTS
```

Workarounds:

(1) If conditions (1), (2), (4)-(A), and (5) are all satisfied,
avoid the symptom in the following way:
If the structure or union is 4 bytes or less in size, add
a dummy member to it so that its size can exceed 4 bytes.

Example:

```
-----------------------------------------------------------------
struct _str {
   long a ;
   char dummy;  // A dummy member added to structure
}              // so that its size can exceed 4 bytes.
```

(2) If conditions (1), (2), (4)-(B), and (5) are all satisfied,
avoid the symptom in the following way:
Allocate the address to the variable using #pragma address.

Example:

```
-----------------------------------------------------------------
#pragma address(OBJB=0xFFFF23)
#pragma pack 1
struct B {
   unsigned short US1:16;
};
struct B OBJB;
void main(){
   OBJB.US1 = 1;
}
```

2.5 With Coding Multi-Loops with the Vacant Loop Elimination Option del_vacant_loop=1 Selected (H8C-081)

Versions Concerned:

V.6.00 Release 00 through V.6.02 Release 00

Symptom:

Loops not empty may be deleted.

Conditions:

This symptom may occur if the following conditions are all satisfied:
(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, AE5, or RS4 is selected (for example, -cpu=2000N in the command line).
(2) As a CPU option, 2000N, 2000A, 2600N, or 2600A is selected, and the Compatibility of output code option -legacy=v4 not selected.
(3) The -optimize=1 option is selected.
(4) The Vacant loop elimination option -del_vacant_loop=1 selected.
(5) In the program exists a multi-loop.
(6) In the controlling expression of the most outside loop exist one

or more expressions delimited each by a comma operator in addition to the continuation conditional expression.

(7) No expressions except reset expressions exist in all the loops inside the loop in (6) and in the loop in (6) itself.

(8) Non of the controlled variables in all the loops inside the loop in (6) and in the loop in (6) itself is referenced after each loop is exited.

(9) In V.6.00 Release 00 through 6.01 Release 02, the types of any variables whose values are incremented in the expressions other than the continuation conditional expression in (6) are different from the type of the controlled variable in the loop in (6).

Example:

```
-------------------------------------------------------------
// -cpu=H8SXA:24
// In a while statement
// -opt=1 and -del_vacant_loop=1
int a, b;
void func_while() {
  int i, j = 0;
  while (a++, b+=2, j < 4){       // Conditions (5), (6)
    for (i = 0; i < 500; i++) {   // Condition (7)
    }
    j++;                          // Condition (7)
  }
   // Condition (8)
}

// -cpu=H8SXA:24
// In a for statement
// -opt=1 and -del_vacant_loop=1
int c, d;
void func_loop() {
  int i, j;
  for (j = 0; c++, d+=2, j < 4; j++) {   // Conditions (5), (6), (7)
    for (i = 0; i < 500; i++) {         // Condition (7)
    }
  }
  // Condition (8)
}

// In V.6.00 Release 00 through V.6.01 Release 02 used.
// In a for statement
```

```
// -opt=1 and -del_vacant_loop=1
long c, d;
void func_loop() {
  int i, j;
  for (j = 0; c++, d+=2, j < 4; j++) {
                                // Conditions (5), (6), (7), (9)
    for (i = 0; i < 500; i++) {      // Condition (7)
    }
  }
  // Condition (8)
}
```

Generated Code:

```
-----------------------------------------------------------------
_func_while
        RTS ; Variables a and b not incremented.
_func_loop
        RTS ; Variables c and d not incremented.
-----------------------------------------------------------------
```

Workarounds:

Avoid the symptom in any of the following ways:

(1) Do not code any loops residing inside the loop in Condition (6) and having no statements within them.

(2) Use the -del_vacant_loop=0 option.

(3) Use the -optimize=0 option.

(4) Place the preprocessor statement "#pragma option nooptimize" immediately before the function where the symptom arises.

2.6 With Putting the Same statement in More Than One Block (H8C-082)

Versions Concerned:

V.6.01 Release 00 through V.6.02 Release 00

Symptom:

If the same statement is put in more than one block,* such statements may be executed with the register without setting a value.

*Here a block is a part of the program enclosed with a pair of braces, { and }.

Conditions:

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

(1) As a CPU option, 300, 300L, 300HN, 300HA, 2000N, 2000A, 2600N,

2600A, H8SXN, H8SXM, H8SXA, H8SXX, AE5, or RS4 is selected
(for example, -cpu=300 in the command line).
(2) The -optimize=1 option is selected.
(3) The same statement is put in more than one block.

Example:

```
------------------------------------------------------------
unsigned char chk0(void){}
unsigned char chk1(void){}
void sub(unsigned char byte){}
void test(void)
{
   if(chk0() == 1 || chk1() == 0){
      sub(0);    // Condition (3)
   }else{
      sub(0);    // Condition (3)
   }            // Then and Else statements are the same.
}
```

Generated Code:

```
------------------------------------------------------------
_test    BSR      @_chk0:8
         CMP.B    #H'01,R0L
         BEQ      @H'000E:8
         BSR      @_chk1:8
         BRA      @_sub:8   ; Parameter of sub not loaded.
```

Workarounds:

Avoid the symptom in any of the following ways:
(1) Use the -optimize=0 option.
(2) Place the preprocessor statement "#pragma option nooptimize"
immediately before the function where the symptom arises.
(3) Place the call to a dummy function or a dummy expression
in the block.

Example:

```
------------------------------------------------------------
// Include function nop() placed.
#include <machine.h>
void test(void)
{
   if(chk0() == 1 || chk1() == 0){
      nop(); // Include function nop() called.
```

```
        sub(0);
    }else{
        sub(0);
    }
}

// A dummy expression placed.
int dummy;
void test(void)
{
    if(chk0() == 1 || chk1() == 0){
        dummy = 0; // A dummy expression placed.
        sub(0);
    }else{
        sub(0);
    }
}
```

2.7 With Assigning the Return Value of a Function to a Member Variable of a Structure, Union, or Class (H8C-083)

Versions Concerned:

V.6.00 Release 00 through V.6.02 Release 00

Symptom:

Return values of functions may incorrectly be assigned to members of a structure, union, or class.

Conditions:

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

(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, AE5, or RS4 is selected (for example, -cpu=2000N in the command line).

(2) As a CPU option, 2000N, 2000A, 2600N, or 2600A is selected, and the Compatibility of output code option -legacy=v4 not selected.

(3) The -optimize=1 option is selected.

(4) A call is made to a function the type of whose return value is 4 bytes or less in length.

(5) The return value in (4) is assigned to a member of a structure. Here the member is 4 bytes long or less of the int or single-precision floating type.

(6) The size of the structure having the member in (5) is 4 bytes or less.

(7) A variable of the structure in (6) is defined as a local variable.

Example:

```
-------------------------------------------------------------
// -cpu=h8sxa
struct ST{           // Condition (6)
   int mem;
}str;
int y;
int sub(void);          // Condition (4)

void temp2(){
   struct ST tmp2;      // Condition (7)
   if (y == 0) {
     tmp2.mem = 3;
   } else {
     tmp2.mem = sub(); // Conditions (4), (5)
   }
   sub2(tmp2.mem);
}
```

Generated Code:

```
-------------------------------------------------------------
_temp2:
        subs    #2,sp
        mov.w   @_y:32,r0
        bne     P_0000000e:8
        bra/s   P_00000014:8
        mov.w   #3:3,r0
P_0000000e:
        jsr     @_sub:24
        mov.w   @sp,r0 ; Return value of sub overwritten.

P_00000014:
        jsr     @_sub2:24
        inc.l   #2,sp
        rts

-------------------------------------------------------------
```

Workarounds:

Avoid the symptom in any of the following ways:

(1) Qualify the local variable in Condition (7) to be volatile.

// Example:

```
void temp2(){
  volatile struct ST tmp2;
```

(2) Define the variables in Condition (7) not as a local variable
but a global one.

// Example:

```
struct ST tmp2; // Variable defined as a global one.
void temp2(){
   if (y == 0) {
```

(3) Use the -optimize=0 option.
(4) Place the preprocessor statement "#pragma option nooptimize"
immediately before the function where the symptom arises.

// Example:

```
#pragma option nooptimze
void temp2(){
  struct ST tmp2;
. . . . . .
   sub2(tmp2.mem);
}
#pragma option
```

3. Two Problems in Assembler

3.1 With Destination Addresses of the jmp Instruction (H8A-0001)

Versions Concerned:

V.4.0 through V.6.02 Release 00

Symptom:

The destination address in the operand of a jmp instruction may incorrectly be specified, resulting in the assembler not functioning properly.

Conditions:

This symptom occurs if condition (A) or condition (B) below is satisfied.
(A) All the conditions from (A-1) to (A-5) below are met.
(A-1) The optimizing option -optimize is selected when the program is assembled.

(A-2) A section in the program area is declared to be in the relative addressing mode (the section hereafter called the relative section).

(A-3) In the above relative section exists an instruction whose code size is modified to a value less than the specified maximum after optimization.*

*An instruction whose code size is modified after optimization

(A-4) Afterward of the instruction in (A-3) exists a local label (in direction of the ascending order of addresses).

(A-5) The local label in (A-4) is referenced by an instruction.

(B) All the conditions from (B-1) to (B-3) below are met.

(B-1) A section in the program area is declared to be in the absolute addressing mode (the section hereafter called the absolute section).

(B-2) Within the above absolute section exists a local label.

(B-3) In an absolute section exists an instruction whose operand is an address operation between a local label and an unresolved address symbol.

Example 1.

```
------------------------------------------------------------
.section sec,code          ; Condition (A-2)
     mov.l   #EQU1,@ER0    ; Condition (A-3)
?LOCAL1                    ; Condition (A-4)
     nop
     mov.l   #?LOCAL1,ER0  ; Condition (A-5)
EQU1:  .equ    4
.end

------------------------------------------------------------
```

Example 2.

```
------------------------------------------------------------
.section sec,code,locate=H'100       ; Condition (B-1)
.import imp_sym
     mov.l #imp_sym + ?LOCAL1, ER0  ; Condition (B-3)
?LOCAL1                         ; Condition (B-2)
     nop
.end

------------------------------------------------------------
```

Workarounds:

Avoid the symptom in any of the following ways:

(A) If condition (A) is satisfied;

(1) Do not use the optimizing option -optimize.

(2) Do not use any local labels.

(3) The first operand of the instruction whose code size is modified by optimization is changed to that of a constant (see Example 1). Or, append to the operand its reserved size (see Example 2).

Example 1.

```
------------------------------------------------------------
.section sec,code
      mov.l   #4,@ER0       ; Changed to a constant.
?LOCAL1
      nop
      mov.l   #?LOCAL1,ER0
EQU1:  .equ    4
.end
------------------------------------------------------------
```

Example 2.

```
------------------------------------------------------------
.section sec,code
      mov.l   #EQU1:8,@ER0    ; Reserved size appended.
?LOCAL1
      nop
      mov.l   #?LOCAL1,ER0
EQU1:  .equ    4
.end
------------------------------------------------------------
```

(B) If condition (B) is satisfied;

(1) Do not use any local labels.

(2) Declare the section in Condition (B-1) to be in the relative addressing mode

3.2 With Placing Labels in an Absolute Section (H8A-0002)

Versions Concerned:

V.4.0 through V.6.02 Release 00

Symptom:

If a label placed in an absolute section is used as the operand of a conditional branch instruction, the branch is made to an incorrect destination after optimization.*

*A section declared to be in the absolute addressing mode

Conditions:

This symptom occurs if the following condition are all satisfied.

(1) The optimizing option -optimize is selected.

(2) A label is placed in an absolute section.

(3) The label in (2) is referenced by an conditional branch instruction in a relative section.

(4) Forward of the label in (2) exists an instruction whose code size is modified to a value less than the specified maximum after optimization.*

*An instruction whose code size is modified after optimization

Example:

```
------------------------------------------------------------
.section rel_sec,code
      bra     ABS_LAB            ; Condition (3)
      nop
.section abs_sec,code,locate=H'200
      mov.l   @(EQU1,ER1),ER0     ; Condition (4)
ABS_LAB:                         ; Condition (2)
      nop
EQU1:  .equ    4
.end
------------------------------------------------------------
```

Workarounds

Avoid the symptom in any of the following ways:

(1) Do not use the optimizing option -optimize.

(2) The first operand of the instruction whose code size is modified by optimization is changed to that of a constant (see Example 1).

(3) To the operand in (2) above, append its reserved size (see Example 2).

(4) Declare the section in which the label referenced by a conditional branch instruction is placed to be in the relative addressing mode.

Example 1.

```
------------------------------------------------------------
.section rel_sec,code
```

```
        bra     ABS_LAB
        nop
.section abs_sec,code,locate=H'200
        mov.l   @(4,ER1),ER0    ; Changed to a constant.
        ABS_LAB
        nop
EQU1:  .equ    4
.end
```

Example 2.

```
.section rel_sec,code
        bra     ABS_LAB
        nop
.section abs_sec,code,locate=H'200
        mov.l   @(EQU1:2,ER1),ER0    ; Reserved size appended.
ABS_LAB
        nop
EQU1:  .equ    4
.end
```

---

4. Three Problems in Optimizing Linkage Editor

4.1 With Calling an Assembler Routine with the Number of Arguments-Storing Registers Being Declared to Be 3 (LNK-0001)

Versions Concerned:

V.4.0 through V.6.02 Release 00

Symptom:

Arguments may incorrectly be passed to an assembler routine in which the number of argument-storing registers is declared to be 3.

Conditions:

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

(1) The compile option -goptimize is selected.

(2) As to the function that makes calls, the conditions from (A) to (C) below are all satisfied.

(A) The function that makes calls does make a call to another within the C/C++ source program.

(B) The arguments of the function that makes calls are stored in two registers.

(C) The arguments of the function that is called by the function in (A) are stored in three registers.

(3) As to the assembler routine and function to be called, conditions (A) and (B) below are satisfied respectively.
(A) Both conditions (A-1) and (A-2) below are met.
(A-1) The assembler routine to be called is defined within the assembler source program.
(A-2) Any of the argument-storing registers ER2, R2, and E2 is referenced as an argument.
(B) Both conditions (B-1) and (B-2) below are met.
(B-1) The function to be called is defined within the C/C++ source program.
(B-2) The function called in (B-1) makes a call to another function, and they make calls to each other.

(4) Any of the following conditions is satisfied to make the optimizing option -optimize=register effective at linking:
(A) The -nooptimize option not selected.
(B) The -optimize=register option selected.
(C) The -optimize option selected with no sub-options used.
(D) The -optimize=safe option selected.
(E) The -optimize=speed option selected.

Example 1.

```
------------------------------------------------------------
--- C source file a.c ----
// ch38 -cpu=h8sxa -goptimize a.c
extern void __regparam3 fasm(long,long,long); // Condition (2)-(C)
long dmy_val;
void jibun();
void fc1(long hiki_dmy){
   fc2();
   dmy_val=hiki_dmy;
}
void fc2(){
   fasm(1,2,3);                    // Condition (2)-(A)
}
--- Assembler source file b.src ---
; asm38 -cpu=h8sxa b.src
   .CPU       H8SXA:24
   .EXPORT    _fasm
   .EXPORT    _val
   .SECTION   P,CODE,ALIGN=2
_fasm:                             ; Condition (3)-(A-1)
```

```
   ADD.L      ER1,ER0
   ADD.L      ER2,ER0                ; Condition (3)-(A-2)
   MOV.L      ER0,@_val:32
   RTS
   .SECTION    B,DATA,ALIGN=2
_val:
   .RES.L     1
   .END

--- Command at linking ---
optlnk -optimize=register a.obj b.obj -start=P,B/400
```

Example 2.

```
--- C source file a.c ----
// ch38 -cpu=h8sxa -goptimize a.c
long dmy,result;
char own(long );
void __regparam3 child(long,long,long ); // Condition (2)-(C)

void loop();

void root (long par ){
   own (par);
   dmy+=par;
}
char own (long par){
   child(1,2,3);                  // Condition (2)-(A)

   dmy = par;
   if(result != 3 )
      return -1;
   else
      return 0 ;
}

void __regparam3 child (long par1,long par2 , long par3){
// Condition (3)-(B-1)
   if(par3 != 3 ){
      loop();                     // Condition (3)-(B-2)
   }
   result = par3;
```

```
}

void loop (){
   child(0,0,0);                    // Condition (3)-(B-2)
}
--- Command at linking ---
optlnk -optimize=register a.obj -start=P,B/400
```

Workarounds:

Avoid the symptom in either of the following ways:

(1) Do not use the optimizing option -optmize=register at linking.

(2) Do not use the Inter-module optimization option -goptimize when compiling the C source file involved (a.c in examples 1 and 2).

4.2 With Referencing the Initial Value of a Variable with the Short Absolute Addressing Mode Option Selected (LNK-0002)

Versions Concerned:

V.4.0 through V.6.02 Release 00

Symptom:

The initial value of a variable may incorrectly be referenced when the Short Absolute Addressing Mode option -optimize=variable_access is selected.

Conditions:

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

(1) The Inter-module optimization option -goptimize is selected to compile C/C++ source modules.

(2) In a source module in (1) exists a variable that is set to its initial value or declared to be of const.

(3) The variable in (2) is of the int or single-precision floating type.

(4) As the initial value of the variable in (2), the address of a function or variable or the one generated by the section address operator is used.

(5) The Address check option (-cpu) is selected at linking, and the ROM area specified by the option and the abs16-declared area are overlaid.

(6) Any of the following conditions is satisfied, so the optimizing option at linking -optimize=variable_access becomes effective:

(A) The -nooptimize option not selected.

(B) The -optimize=register option selected.

(C) The -optimize option selected with no sub-option used.
(D) The -optimize=speed option selected.
(E) The -optimize=variable_access option selected.

Example:

```
-----------------------------------------------------------------
--- C source file a.c ----
// ch38 -cpu=2600a -goptimize a.c
long val;
long val_addr = (long)&val;
char func(){
  if (val_addr == (short)&val )
. . . . . .
}
--- Command at linking ---
optlnk -optimize=variable_access a.obj -cpu=ROM=0-7fff,ROM=10000-11000
-start=P,D,B/10000
-----------------------------------------------------------------
```

Workarounds:

Avoid the symptom in any of the following ways:
(1) Use the label of the variable in Condition (2) in the
Optimization partially disabled option -variable_fobid at linking.
(2) Do not use the optimizing option -optimize=variable_access
at linking.
(3) Use the optimizing option -nooptimize at linking.
(4) Do not use the Inter-module optimization option -goptimize when
compiling the C source file involved (a.c in the above example).

4.3 With Using the Same Code Unification Option (LNK-0003)

Versions Concerned:

V.4.0 through V.6.02 Release 00

Symptom:

When the Same code unification option is effective, references to character strings and assignments to variables of the double-precision floating type may incorrectly be made.

Conditions:

This symptom may occur if the following conditions are all satisfied:
(1) The Inter-module optimization option -goptimize is selected to
compile C/C++ source modules.

(2) In a source module in (1) exists a reference to a character string or assignment to a variable of the double-precision floating type, or in the code generated by the compiler exists an instruction that adds a constant to a variable's address to access memory.
(3) Any of the conditions from (A) to (D) below is met.
(A) The Same code unification option -optimize=same_code and the Uses short absolute addressing mode option -variable_access selected.
(B) The Speed optimization option -optimize=speed selected.
(C) The -optimize option is selected with no sub option used.
(D) The -nooptimize option not selected.

Example:

```
----------------------------------------------------------------
--- C source file a.c ----
// ch38 -cpu=h8sxa -goptimize -op=0 tp1.c

short check( char* a, long dmy_para){
  if( *a ==’s’) return 0;
  return 0;
}
short func1(){
  char *p =”string”;
  return check(p,0xABCDEF);
}
short func2(){
  char *p=”string”;
  return check(p,0xABCDEF);
}
short func(){
  if( 0 == func1() ) return 0;
  return 1;
}

--- Command at linking ---
optlnk -optimize=variable_access,same_code -cpu=ROM=0-7fff,
ROM=10000-11000 -start=P,D,B/10000 a.obj
----------------------------------------------------------------
```

Workarounds:

Avoid the symptom in any of the following ways:
(1) Do not use the optimizing option -optimize=same_code at linking.

(2) Use the optimizing option -nooptimize at linking.
(3) Do not use the Inter-module optimization option -goptimize when compiling the C source file involved (a.c in the above example).

5. Schedule of Fixing the Problems

We plan to fix these problems in the next release of the product.
