RENESAS Tool News

RENESAS TOOL NEWS on July 16, 2005: RSO-H8C-050716D

Notes on Using the C/C++ Compiler Package V.6 for the H8SX, H8S, and H8 Families of MCUs

Please take note of the eight problems described below in using the C/C++ compiler package for the H8SX, H8S, and H8 families of MCUs.

1. Versions Concerned

C/C++ compiler package V.6.00 Release 00 through V.6.00 Release 03, and V.6.01 Release 00

2. Problems

2.1 On an Expression One of Whose Operand Is Not a Number (H8C-0017)

If an equality expression or an additive one between a variable and not a number (NAN) is evaluated, an incorrect result may be obtained.

Conditions:
This problem occurs if the following conditions are all satisfied:

1. Any of the options cpu=2000n, 2000a, 2600n, 2600a, h8sxn, h8sxm, h8sxa, and h8sxx is selected.
2. The following equality expression (evaluation of equality or inequality) in an if statement or additive expression (addition or subtraction) is evaluated:
   - if (variable == NAN) or if (variable != NAN) (float type)
   - variable + NAN or variable - NAN (both float or double type)

Example:

```
-------------------------------------------------------
float x,y;
```

```
const float z_nan = 0.0/0.0;
void test() {
    x = y - z_nan;   /* Condition 2 */
                 /* Interpreted as x=y
                    if not a 0 but NAN subtracted */
}
-------------------------------------------------------
```

Workaround:
Delete the definition of NAN in the file and define it as a variable in another file; then compile it.

Modification of Example above:

```
-------------------------------------------------------
float x,y;
extern const float z_nan;  /* Define z_nan in a separate file
*/
void test() {
   . . . . . . . . . . . . . . . . . . . . . . . .
/* Separate file */
const float z_nan=0.0/0.0;
-------------------------------------------------------
```

2.2 On Assigning Values to Members of a Structure (H8C-0018)
If a structure whose size is 4 bytes or less is defined and then the variables of this structure type are declared within a function, the assignment of values to members of the structure may not be performed.

Conditions:
This problem occurs if the following conditions are all satisfied:

1. Any of the options cpu=2000n, 2000a, 2600n, 2600a, h8sxn, h8sxm, h8sxa, and h8sxx is selected.
2. The optimize=1 option is selected.
3. A structure is defined; then the variables of this structure type are declared within a function.
4. The structure in Condition 3 is 4 or 3 bytes in size and its members are defined in any of the following orders (where signed and unsigned are omitted; that is, int or short can be signed or unsigned).
   char, char, char, char
   char, char, int or short
   char, char, char

char, short or int, char (if the pack=1 option selected.)

5. The return value of a function is assigned to a char-type member of the structure in Condition 3.
6. The structure-type variables in Condition 3 are not qualified to be volatile.
7. The following registers for storing arguments are assigned to two or more structure-type variables in Condition 3:
   - Registers ER0 and ER1: If the -regparam=2 option used.
   - Registers ER0, ER1, and ER2: If the -regparam=3 option used.

Example:

```
-----------------------------------------------------
struct _str {                  /* Condition 4 */
   char  m_c1;
   char  m_c2;
   char  m_c3;
   char  m_c4;
};
extern char sub();
void func() {
    struct _str str1;           /* Conditions 3 and 6 */
    struct _str str2;           /* Conditions 3 and 6 */
    :
    str1.m_c3 = sub();           /* Condition 5 */
                    /* No value assigned to str1.m_c3 */
    :
    str2.m_c1 = sub();           /* Condition 5 */
    :
}
-----------------------------------------------------
```

Workaround:
This problem can be circumvented in any of the following ways:

(a) Use the optimize=0 option.

(b) Qualify the structure-type variables to be volatile.

Modification of Example above:

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

```
volatile struct _str str1;
volatile struct _str str2;
-----------------------------------------------------
---
```

(c) Make the size of the structure greater than 4 bytes.

Modification of Example above:

```
-----------------------------------------------------
---
struct _str {
   char  m_c1;
   char  m_c2;
   char  m_c3;
   char  m_c4;
   char  dummy;    // A dummy member added
};
-----------------------------------------------------
---
```

2.3 On Assigning Values to Elements of an Array (H8C-0019) If values are assigned to elements of an array before an if statement and in its Then statement, the assignments may not correctly be performed.

Conditions:
This problem occurs if the following conditions are all satisfied:

1. Any of the options cpu=2000n, 2000a, 2600n, 2600a, h8sxn, h8sxm, h8sxa, and h8sxx is selected.
2. The optimize=1 option is selected.
3. A selection statement (an if statement with no else one) exists in an iteration statement.
4. Values are assigned to elements of an array before an if statement and in its Then statement in Condition 3.
5. When the right term of the assignment expression before the if statement is called Expression 1, and the one in the Then statement Expression 2, the types of Expressions 1 and 2 are different from one the other.
6. Expressions 1 and 2 in Condition 5 satisfy either of the following:
   - Expression 1 may takes a value that is out of the range expressible in Expression 2.

- Expression 2 may takes a value that is out of the range expressible in Expression 1.

Example:

```
-------------------------------------------------------
#define MAX 10
int A[MAX];
 int x, y, z;
 char sc0, sc1, sc2;
 test(){
   int i;
   for(i = 0; i < MAX; i++){
       A[i] = x + y;          /* Conditions 4, 5, and 6 */
       /* When if(z) is FALSE, x+y is incorrectly
                                  assigned to array A[i] */
       if(z){                 /* Condition 3 */
           A[i] = sc0;        /* Conditions 4, 5, and 6 */
       }
       x++;
       y++;
       sc0++;
    }
}
-------------------------------------------------------
```

Workaround:
This problem can be circumvented in any of the following ways:

(a) Use the optimize=0 option.

(b) Place an nop() include function in the then statement.

Modification of Example above:

```
----------------------------------------------------------
#include <machine.h>
 . . . . . . . . . . . . . . . . . . . . . . . .
 if(z){
 nop(); // An include function placed
   A[i] = sc0;
}
----------------------------------------------------------
```

(c) Introduce an else statement, in which place an nop() include function.

Modification of Example above:

```
---------------------------------------------------
---
#include <machine.h>
 . . . . . . . . . . . . . . . . . . . . . . . .
 if(z){
 A[i] = sc0;
 } else{
 nop();          /* An include function placed */
 }
---------------------------------------------------
---
```

(d) Place an nop() include function between Expression 1 and the if statement.

```
---------------------------------------------------
---
#include  <machine.h>
 . . . . . . . . . . . . . . . . . . . . . . . .
  A[i] = x + y;
  nop();           /* An include function placed */
  if(z){
      A[i] = sc0;
  }
---------------------------------------------------
---
```

2.4 On Placing User Labels in the __asm include assemble function (H8C-0020) When the __asm{} include assemble function used, an incorrect area may be accessed if the number of symbols (functions, variables, and labels) is equal to or greater than 256.

Conditions:
This problem occurs if the following conditions are all satisfied:

1. Any of the options cpu=2000n, 2000a, 2600n, 2600a, h8sxn, h8sxm, h8sxa, and h8sxx is selected.
2. An include assemble function __asm{} is used in the

program.

3. Labels are defined in the __asm{} function.
4. The number of symbols (functions, variables, and labels) is equal to or greater than 256.
5. Among the symbols in Condition 4 exist variables that are output to an assembler file by the assembler directive command .EXPORT. (To see whether those variables exist or not, compile the program with the -code=asm option selected. Then check the assembly program generated.)
6. Several expressions contain symbols described in Condition 5.

Example:

```
-------------------------------------------------------
int x000,x001,x002,x003,x004,x005,x006,x007,x008,x009;
 . . . . . . . . . . . . . . . . . . . . . . . . . .
int x250,x251,x252,x253,x254,x255;        /* Condition 4 */
void f1(){
   x013=10;                        /* Condition 6 */
          /* If x013 not exported, this write is made
                              to an incorrect area */
   __asm{                          /* Condition 2 */
      label1;                      /* Condition 3 */
      label2;
   }
}
-------------------------------------------------------
```

Workaround:
Split a file into several ones so that the number of symbols may become less than 256. Then compile those files.

2.5 On Missing Data Width at Generating Assembler Source Files (H8C-0021)

If (1) the -code=asmcode option is selected, (2) assembly codes are included, and (3) the BRA/BC and BRA/BS instructions are used in absolute addressing mode, values representing bit width (n in @aa:n) may not be generated.

Conditions:
This problem occurs if the following conditions are all satisfied:

1. Any of the options cpu=h8sxn, h8sxm, h8sxa, and h8sxx is selected.
2. The code=asmcode option is selected.
3. Pragma preprocessing directive #pragma asm--#pragma endasm or #pragma inline_asm is used for including assembly codes.
4. In the C source program exist statements that are converted to BRA/BC and BRA/BS instructions.
5. The BRA/BC and BRA/BS instructions are expressed in absolute addressing mode.

Example:

```
------------------------------------------------------
struct ST {
   unsigned char HH:1;
   unsigned char HL:1;
};
#define STR (*(volatile struct ST *)0xFFFFA1)
volatile int dd;
void func(){
   if(STR.HL){
      return;
   }
 /***** Incorrect code generated *****/
  BRA/BS     #6,@H'A1,L22
 /***** Correct code to be generated *****/
  BRA/BS     #6,@H'A1:8,L22
   dd = 0;
   return;          #pragma asm
   ;SLEEP
   #pragma endasm
}
------------------------------------------------------
```

Workaround:
Move the included assembly codes to a separate file; then compile it.

2.6 On Overwriting Values in Registers (H8C-0022)

If a function uses all the registers (ER0--ER7), and values of the function are loaded in the registers assigned to them, the values already stored in these registers may be overwritten.

NOTE: This problem arises in V.6.01 Release 00 only.

Condition:
This problem occurs if the following conditions are all satisfied:

1. Any of the options cpu=2000n, 2000a, 2600n, 2600a, h8sxn, h8sxm, h8sxa, and h8sxx is selected.
2. All the registers, ER0--ER7, are used for storing generated codes.
3. There are no vacant registers that can be assigned to variables.

Example:

```
-------------------------------------------------------
  . . . . . . . . . . . . . . . . . . . . . . . .
void sub1(STRUCT *p1,STRUCT *p2,STRUCT *p3,STRUCT
*p4,
STRUCT *p5,STRUCT *p6,STRUCT *p7){
   p7->data=pfunc1(p1->data,p2->data,p3->data,
   p4->data,p5->data,p6->data) +
 /***** Generated Codes *****/
  . . . . . . . . . . . . . . . . . . . . . . . .
  MOV.L      @_pfunc1:32,@(H'0010:16,SP)
  MOV.B      @(H'000F:16,SP),R0L
  MOV.B      R5H,R0H
  MOV.B      R5L,@(9:16,SP)
  MOV.B      R4H,@(8:16,SP)
  MOV.B      R4L,R2L
  MOV.B      R3H,R2H
 /***** Incorrect code generated *****/
  MOV.L      @(H'0010:16,SP),ER1; Value in register for
storing
  JSR        @ER1            ; function's argument overwritten
 /***** Correct code to be generated *****/
  MOV.B      R5L,R1L
  MOV.B      R4H,R1H
  MOV.L      @(H'0010:16,SP),ER6
  JSR        @ER6
-------------------------------------------------------
```

Workaround:
Sorry that there is no solution for this problem.

2.7 On Incorrect Positions of Labels in Assembly Files (H8C-0023)

If programs containing delayed jump instructions are compiled, labels may be put at incorrect positions in assembly files.

Conditions:
This problem occurs if the following conditions are all satisfied:

1. Any of the options cpu=h8sxn, h8sxm, h8sxa, and h8sxx is selected.
2. The optimize=1 option is selected.
3. The code=asmcode option is also selected.
4. Labels exist after an instruction* next to a delayed jump instruction
   *: The RTE, RTE/L, and SLEEP instructions excluded.

Example:

```
-------------------------------------------------------
typedef unsigned char TYPE;
extern char val;
void sub03(void ){
   char ans=0;

   switch(val){
      case  0: ans=0; break;
      case  1: ans=1; break;
      case  3: ans=3; break;
   }
   if(ans==3) sub();       /* Always FALSE as always ans=1
*/
}
-------------------------------------------------------
```

Workaround:
Use either the optimize=0 or code=machinecode option.

2.8 On Placing a Variable at an Odd Address (H8C-0024)

If the address boundary of a member of a structure-type variable is adjusted by 2 bytes, the member may be placed at an odd address.

Conditions:

This problem occurs if the following conditions are all satisfied:

1. Any of the options cpu=2000n, 2000a, 2600n, 2600a, h8sxn, h8sxm, h8sxa, and h8sxx is selected.
2. A structure whose size is shown below is defined.

Windows edition:	0x100000 bytes or more and 0x1FFFFE bytes or less
UNIX edition:	0x10 bytes or more and 0x1E bytes or less

3. In the structure in Condition 2 exist members whose types are other than signed and unsigned char.
4. The noalign option is selected; or the optimize=variable_access option* is selected at linking.
5. The variables having the type of the structure in Condition 2 are defined between the definitions of odd-sized variables.
   NOTE:
   *: When the goptimize option is used for more than one file, this problem may occur if the optimization using the short absolute addressing mode is performed at the same time.

Example:

```
----------------------------------------------------
typedef unsigned char TYPE;
 struct ST {                /* Condition 2 */
   char   c1;
   char   c2[12];
   int    d1;             /* Condition 3 */
 } ;
 char dummy1;               /* Condition 5 */
 struct ST east1[2];        /* Member d1 placed at odd
address */
 char dummy2;               /* Condition 5 */
 struct ST est1;
 char dummy3;
 struct ST est2;

 struct ST  sub6()
 {
    struct ST  *pst1;
    pst1 = &east1[1];
```

```
     *--pst1 = est1;
     return(*pst1);
 }
--------------------------------------------------------
```

Workaround:
Don't use the noalign option at compilation and the ptimize=variable_access option at linking.

3. **Schedule of Fixing the Problem**

We plan to fix the above eight problems in the V.6.01 Release 01.
