RENESAS Tool News

RENESAS TOOL NEWS [July 1, 2003: RSO-M3T-CC32R-030701D]

A Note on Using Cross-Tool Kit M3T-CC32R

Please take note of the following problem in using the M3T-CC32R cross-tool kit for the M32R family MCUs:

- On converting the address of a variable to a pointer whose type is different from the variable's and performing an indirect operation on it.

---

1. **Versions Concerned**
   M3T-CC32R V.4.00 Release 1 and V.4.10 Release 1

2. **Description**
   When the address of a variable is converted to a pointer whose type is different from the variable's, and the C source file that performs an indirect operation on the pointer is compiled, incorrect code will be generated or the following error message be displayed (see the note at the end of Section 2.1):

   cg32R: "...", line xx: internal error: illegal IL, illegal lhs of ASSIGN

   Here, a filename and line number appear in the double quotes and in place of XX respectively.

   2.1 Conditions

   This problem occurs if the following six conditions are satisfied:

   (1) An optimizing option covering the -O4 option's function is used at compilation. (Any of the -O4, -O5, -O6, and -O7 options is selected; or the -Otime or -Ospace option only is used.)

   (2) There exists an int-, floating- or pointer-type variable.

   (3) The address of the variable in (2) is cast to a pointer whose type is different from the variable's.

   (4) An operation is performed on the pointer in (3) using any of the following indirect operators:

   (a) the indirect operator (*)

(b) the indirect member operator (->) for a structure; member this operator specifies needs to exist in the forefront of the structure.

(c) the indirect member operator (->) for a union

(5) The variable in (2) and the object of any operator in (4) are the same in size, different in type, and are not qualified as volatile.

(6) The object of any operator in (4) is used in any of the following eight terms, operands, etc., (a) through (h), and if used in any of the last four, (e) through (h), either this object or the variable in (2) must be a float type:

(a) in the left term of an assignment operator (=) provided that no other operators are used

(b) in the operand of an increment or decrement operator (++ or --) provided that no other operators are used

(c) in the right term of an assignment operator (=) provided that no other operators are used; however, if the left and right terms of the operator are different in type, either the object or the variable in (2) must be a float type

(d) in the 4-byte operand at the right side of any binary operator except assignment and direct/indirect member operators (=, ->, and .)

(e) in the operand at the left side of any binary operator except assignment and member operators

(f) in the operand of any prefix operator except & and sizeof

(g) in the controlling expression, continuation conditional expression, or any of the other possible expressions except the assignment expression in such a control statement as for, while, do, if, switch, or return

(h) in an actual argument of any function

NOTE:
In (a), (b), and (c) of Condition (6), if the left and right terms of the operator are the same in type, the error message shown in Section 2 will be displayed; if they are different in type, or any of the sub-conditions (d) through (h) is met, incorrect code be generated.

2.2 Examples

Source file: sample1.c

```
------------------------------------------------------------
  long func1(void)
  {
     long sl;                    /*Conditions (2)and(5)*/
```

```
    unsigned long *ptr_ul;
    ptr_ul = (unsigned long *) &sl; /*Condition (3)*/
    *ptr_ul = 0x80000000;          /*Conditions (4)(a),(5),
                                   and (6)(a)*/
    return sl;
  }
-----------------------------------------------------------------
```

Source file: sample2.c

```
-----------------------------------------------------------------
  signed char sc;                  /*Conditions (2)and(5)*/
  unsigned char uc;
  struct uc_str {
       unsigned char  uc;
  };
  void
  func2(void)
  {
    struct uc_str  *ptr_ucs;
    ptr_ucs = (struct uc_str *) &sc; /*Condition (3)*/
    uc = ptr_ucs->uc;              /*Conditions (4)(b),(5),
                                   and (6)(c)*/
  }
-----------------------------------------------------------------
```

Source file: sample3.c

```
-----------------------------------------------------------------
  long s1, s2;
  void
  func3(unsigned long ul)       /*Conditions (2)and(5)*/
  {
    long *ptr_sl = (long*)&ul; /*Condition (3)*/
    s1 = s2 + *ptr_sl;         /*Conditions (4)(a),(5),
                                   and (6)(d) */
  }
-----------------------------------------------------------------
```

Source file: sample4.c

```
-----------------------------------------------------------------
  long
  func4(float f)               /*Conditions (2)and(5)*/
  {
    long *ptr_sl;
```

```
    ptr_sl = (long *) &f;      /*Condition (3)*/
    return *ptr_sl;           /*Conditions (4)(a),(5),
                                  and (6)(g)*/
  }
```
---

3. **Workaround**

This problem can be circumvented in either of the following ways:

(1) Don't use any optimizing option covering the -O4 option's function (-O4, -O5, - O6 and -O7).
When using -Otime or -Ospace, select any of the -O0, -O1, -O2, and -O3 options at the same time.

(2) Qualify the variable in Condition (2) or the object of an operator in Condition (4) as volatile.

Modified Examples

Source file: sample1.c (modified)

```
---------------------------------------------------------------
  long func1(void)
  {
     long sl;
     volatile unsigned long *ptr_ul;   /* Qualified as volatile */
     ptr_ul = (volatile unsigned long *) &sl;
     *ptr_ul = 0x80000000;
     return sl;
  }
---------------------------------------------------------------
```

Source file: sample2.c (modified)

```
---------------------------------------------------------------
  volatile signed char sc;            /* Qualified as volatile */
  unsigned char uc;
  struct uc_str {
       unsigned char  uc;
  };
  void
  func2(void)
  {

     struct uc_str  *ptr_ucs;
     ptr_ucs = (struct uc_str *) &sc;
     uc = ptr_ucs->uc;
```

```
  }
-----------------------------------------------------------------

Source file: sample3.c (modified)
-----------------------------------------------------------------

  long s1, s2;
  void
  func3(volatile unsigned long ul)     /* Qualified as volatile */
  {
     long *ptr_sl = (long*)&ul;
     s1 = s2 + *ptr_sl;
  }
-----------------------------------------------------------------

Source file: sample4.c (modified)
-----------------------------------------------------------------
  long
  func4(float f)
  {
     volatile long *ptr_sl;          /* Qualified as volatile */
     ptr_sl = (volatile long *) &f;
     return *ptr_sl;
  }
-----------------------------------------------------------------
```

4. **Schedule of Fixing the Problem**

   We plan to fix this problem in our next release of the product.
