RENESAS Tool News

MAEC TOOL NEWS: MAECT-M3T-NC308WA-030201D

Notes on Using C Compilers M3T-NC308WA and M3T-NC30WA

Please take note of the following problem in using C compilers (with an assembler and integrated development environment) M3T-NC308WA and M3T-NC30WA:

- On hexadecimal escape sequences of string literals
- On subtract operations of addresses
- On conditional jumps in iteration control structures

---

1. Problem on Hexadecimal Escape Sequences of String Literals

1.1 Products and Versions Concerned

M3T-NC308WA V.5.00 Release 1

for the M32C/80 and M16C/80 series MCUs

M3T-NC30WA V.5.00 Release 1 and Release 2

for the M16C/60, M16C/30, M16C/20, and M16C/10 series MCUs

1.2 Description

If characters represented by hexadecimal escape sequences (Note 1) exist in a string literal (Note 2), they may not be replaced with the correct characters at compilation. However, note that character constants (Note 3) represented by hexadecimal escape sequences can be replaced with the correct ones.

Notes:
1. Hexadecimal numbers following ¥x or ¥X
2. A character string enclosed with double quotes
3. Characters enclosed with single quotes

1.3 Conditions

This problem occurs if the following two conditions are satisfied:

(1) Hexadecimal escape sequences are used in a string literal.
(2) Any of the hexadecimal numbers in the hexadecimal escape sequences in (1) above contains one or more of the lowercase letters a, b, c, d, e, and f.

1.4 Example

```
---------------------------------------------------------------------
  const char s[] = "¥x5a¥x0d¥x0a";
                /*'Z', control character CR, control character LF*/
---------------------------------------------------------------------
```

Here, you will obtain 81H, 34H, and 31H instead of the correct three bytes of 5AH, 0DH, and 0AH.

1.5 Workaround

This problem can be circumvented in any of the following ways:
(1) Use uppercase letters in hexadecimal escape sequences.
(2) Use simple escape sequences.
(3) Use octal escape sequences.
(4) Use character constants or hexadecimal integer constants to initialize arrays.

```
---------------------------------------------------------------------
  const char s1[] = "¥x5A¥x0D¥x0A";  /* Uppercase letters in
                              hexadecimal escape sequences */
  const char s2[] = "Z¥r¥n";           /* Simple escape sequences */
  const char s3[] = "¥132¥015¥012";     /* Octal escape sequences */
  const char s4[] = { 'Z', 0x0d, 0x0a }; /* A character constant and
                              hexadecimal integer constants */
---------------------------------------------------------------------
```

1.6 Schedule of Fixing the problem

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

---

2. Problem on Subtract Operations of Addresses

2.1 Product and Version Concerned

M3T-NC308WA V.5.00 Release 1 for the M32C/80 and M16C/80 series MCUs

2.2 Description

When a C-language source program contains an expression subtracting the address of a variable or function from another variable, compiling the program may result in

"System Error" being displayed.

2.3 Conditions

This problem occurs if the following four conditions are satisfied:
(1) In the source program exists an expression including a subtraction.
(2) The result of the subtraction is stored in a variable on memory (including the case where the result is stored in the area indirectly referenced by a pointer).
(3) The minuend is a variable on memory (including the case where a variable is indirectly referenced by a pointer).
(4) The subtrahend is any of the following or the type-cast values of them:

- The address of a variable (or the name of an array) declared outside of a function
- The address of a variable (or the name of an array) declared inside a function using the static qualifier
- The address of a function
- A pointer to which any of the above three is assigned

2.4 Example

```
-----------------------------------------------------------------------
  extern int  iArr[10];         /* Condition (4) */
  extern long num;               /* Condition (2) */

  void func(long p)             /* Condition (3) */
  {
     num = p - (long)&iArr[0];
                          /* Conditions (1),(2),(3), and (4) */
  }
-----------------------------------------------------------------------
```

2.5 Workaround

Create a temporary variable in a function and store the subtrahend in it.
Next, place a dummy asm function immediately after storing the subtrahend. Then, perform the subtraction using the temporary variable.

```
-----------------------------------------------------------------------
  void func(long p)
  {
     long tmp = (long)&iArr[0];  /* The subtrahend stored
                                    in the temporary variable. */
     asm();                   /* A dummy asm function placed */
     num = p - tmp;             /* The subtraction performed */
  }
-----------------------------------------------------------------------
```

2.6 Schedule of Fixing the problem

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

---

3. Problem on conditional jumps in Iteration Control Structures

3.1 Products and Versions Concerned

M3T-NC308WA V.5.00 Release 1

for the M32C/80 and M16C/80 series MCUs

M3T-NC30WA V.5.00 Release 1 and Release 2

for the M16C/60, M16C/30, M16C/20, and M16C/10 series MCUs

3.2 Description

Code for conditional jumps in iteration control structures may not be created.

3.3 Conditions

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

(1) Any one or more of the options -O, -O[1-5], -OR, and -OS are used.

(2) A function contains an iteration control structure.

(3) In the function in (2) is declared a variable that is not static but of type char, unsigned char, unsigned int, or unsigned long; or a pointer.

(4) Before the iteration processing, a constant (hereafter called the constant "a") is assigned to the variable in (3).

(5) The variable in (3) is compared with another constant (hereafter called the constant "b") using the "==" or "!=" operator in the iteration control structure.

(6) The value of the most significant bit of the constant "a" is different from that of the constant "b".

(7) In the iteration control structure, a constant is subtracted from the variable in (3) with the value of the most significant bit of the constant "a" being 1, or a constant is added to the variable in (3) with the value of the most significant bit of the constant "a" being 0.

3.4 Examples

[Example 1]

```
------------------------------------------------------------------------
  int x, y;
  void func(void)
  {
     unsigned int i;                    /* Condition (3) */

     for (i = 0x8000U; i != 0x7fffU; i--) { /* Conditions (2),(4),
                                          (5),(6),and(7) */
        x += y;                       /* Constant "a" = 0x8000U */
     }                               /* Constant "b" = 0x7fffU */
  }
------------------------------------------------------------------------
[Example 2]
------------------------------------------------------------------------
  int func2(void)
  {
     int near    *p = (int near *)0x7000;  /* Conditions (3)and(4) */
                                  /* Constant "a" = 0x7000 */
     int         sum = 0;

     while (p != (int near *)0x8000) {     /* Conditions (2),(5),
                                             and (6) */
                                  /* Constant "b" = 0x8000 */
        sum += *p;
        p++;                         /* Condition (7) */
     }
     return sum;
  }
------------------------------------------------------------------------
[Example 3]
------------------------------------------------------------------------
  void func3(void)
  {
     char c = 0x80;                    /* Conditions (3)and(4) */
                                  /* Constant "a" = 0x80 */
  TOP:                               /* Condition (2) */
     c--;                           /* Condition (7) */
     if (c == 0x7f) {                  /* Conditions (5)and(6) */
                                  /* Constant "b" = 0x7f */
        a = 0;
        return;
     }
     b++;
```

```
    goto TOP;
  }
```

3.5 Workaround

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

(1) At every comparison where the code is not generated, use any of the four operators "<=", ">=", "<", and ">" instead of "!=" and "==".

(2) If none of the above four operators can be used (in Example 3, for instance), place a dummy asm function between comparison and addition; or between comparison and subtraction.

[Sidestepping of Example 1]

```
  int a, b;
  void func(void)
  {
    unsigned int i;

                          /* Replace "!=" with ">" */

    for (i = 0x8000U; i > 0x7fffU; i--) {

      a += b;
    }
  }
```

[Sidestepping of Example 2]

```
  int func2(void)
  {
    int near    *p = (int near *)0x7000;
    int         sum = 0;

                          /* Replace "!=" with "<" */
    while (p < (int near *)0x8000) {
      sum += *p;
      p++;
    }
    return sum;
  }
```

[Sidestepping of Example 3]

```
------------------------------------------------------------------------
  void func3(void)
  {
     char c = 0x80;
  TOP:
     c--;
     asm();                    /* Place asm(); here */
     if (c == 0x7f) {
        a = 0;
        return;
     }
     b++;
      goto TOP;
  }
------------------------------------------------------------------------
```

3.6 Schedule of Fixing the Problem

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