RENESAS Tool News

RENESAS TOOL NEWS on January 28, 2014: 140128/tn1

Note on Using C compiler for RL78 Family and 78K0R of MCUs (CA78K0R), C compiler for 78K0R of MCUs (CC78K0R) and Assembler for 78K0R of MCUs (RA78K0R)

When using the C compiler for the RL78 family and 78K0R of MCUs (CA78K0R), the C compiler for 78K0R of MCUs (CC78K0R) and the assembler for 78K0R of MCUs (RA78K0R), take note of the following problems:

- With incorrect code being output for the processing of multiple casts of floating-point constants
- With incorrect code being output for processing of the near/far qualifier for array pointers
- With incorrect code being output from multiplication, division, remainder arithmetic, and indirect reference expressions
- With the assert function not operating normally
- With return of an error when a one-bit-wide bit field is used in a Conditional Expression
- With the conversion of character strings by the strtol and strtoul functions producing incorrect numerical values
- With incorrect address reference being made in the case of reference to a symbol resolved by the assembler for the RL78-S1 core
- With incorrect code being output for the CALL directive
- With the BR and CALL directives producing errors

---

1. Problem with Incorrect Code Being Output for the Processing of Multiple Casts of Floating-Point Constants

1.1 Products and Versions Concerned

CA78K0R V1.20 to V1.60
(included in the integrated development environment CubeSuite+)
CA78K0R V1.00 to V1.10
(included in the integrated development environment CubeSuite)
CC78K0R V1.00 to V2.13
(bundled with the integrated development environment PM+)

1.2 Description

Multiple casting of floating-point constants produces incorrect results of operations.

1.3 Conditions

This problem arises if the following conditions are all met:

(1) A floating-point constant or constant cast to a floating-point type is cast to a floating-point type.

(2) The constant described in condition (1) above is cast to an integer type.

(3) The result of the operation described by condition (2) above is used in an operation other than the following:

- Simple assignment operation: =
- Logical operation: && or ||
- Conditional operation: ? :
- unary operation: !

1.4 Example

```
------------------------------------------------------------------
[*.C]
#define A ((long)((double)6031.0))  // Conditions (1) and (2)
void func(void)
{
   long x;
   x=A<<1;                  // Condition (3)
}
------------------------------------------------------------------
```

The result is not x = 12062 (= 6031 * 2).

1.5 Workaround

Do not cast a floating-point constant or constant cast to a floating-point type to a floating-point type.

```
------------------------------------------------------------------
#define A ((long)6031.0)
void func(void)
{
   long x;
   x=A<<1;
}
------------------------------------------------------------------
```

The result is x = 12062 (= 6031 * 2), which is correct.

2. Problem with Incorrect Code Being Output for Processing of the Near/Far Qualifier for Array Pointers

2.1 Products and Versions Concerned

CA78K0R V1.20 to V1.60
(included in the integrated development environment CubeSuite+)
CA78K0R V1.00 to V1.10
(included in the integrated development environment CubeSuite)
CC78K0R V1.00 to V2.13
(bundled with the integrated development environment PM+)

2.2 Description

Incorrect code may be output because the near or far qualifier is not effective on the array pointer at the point where the type name in the expression is described. In addition, in that case, a warning message may not be correctly output.

2.3 Conditions

This problem arises if any of the following conditions are met:

Condition 1:

Incorrect code is output when all three conditions listed below are met. In addition, in that case, a warning message may not be correctly output.

(1) The small model (-ms option) or the medium model (-mm option) of the compiler is used, or the type of the memory model is not changed. (see NOTE)
(2) A pointer to a far array is used as the type name in an operand for the cast operator.
(3) A far address is included in the operand cast in item (2) above.

NOTE:
If the type of the memory model is not changed, the problem arises because the medium model is selected by default.

Example for condition 1:

```
------------------------------------------------------------------
[*.C]
typedef __far int *P;
typedef __far int (*PA)[10];

P p2;
PA pa1, pa2;
int i1;
void func(void)
{
 . . . . . .
   pa2 = (__far int (*)[10])p2;       // (2), (3) and
                                      // the following (a)
```

```
    i1 = (*(__far int (*)[10])pa1)[0];  // (2), (3) and
                                        // the following (b)
. . . . . .
}
```

In the example above, (int (*)[10]) is used in the cast because the far qualifier is not effective. Thus, it is interpreted as (__near int (*)[10]) in the small and medium models and, as a result, incorrect code is output.

(a) The warning message below is output and incorrect code is output due to the mismatch between the sizes of the pointers on the right and left sides as the right side did not include a cast to _far.

Warning:

CC78K0R warning W0416: Illegal type and size (far/near) pointer combination

---

Example of code output for (a) under condition 1:

```
; line    X :     pa2 = (__far int (*)[10])p2;
;       Incorrect code                  Correct code
        movw ax,!_p2                    ; movw ax,!_p2
        mov  _@SEGAX,#0FH ; 15   ;
        cmpw ax,#00H            ; 0     ;
        sknz                              ;
        clrb _@SEGAX                    ;
?L0004:                                 ;
        movw !_pa2,ax                   ; movw !_pa2,ax
        mov  a,_@SEGAX                  ; mov  a,!_p2+2
        mov  !_pa2+2,a                  ; mov  !_pa2+2,a
```

(b) The incorrect code is output without a warning message as the types of the right and left sides match due to the indirect reference on the right side.

Example of code output for (b) under condition 1:

```
; line    xx :     i1 = (*(__far int (*)[10])pa1)[0];
;       Incorrect code            Correct code
        movw de,!_pa1             ; movw de,!_pa1
                                    ; mov  a,!_pa1+2
                                    ; mov  ES,a
        movw ax,[de]              ; movw ax,ES:[de]
```

```
        movw !_i1,ax            ; movw !_i1,ax
  ----------------------------------------------------------------
```

Condition 2:

Incorrect code is output when both of the conditions below are met:

(1) The small model (-ms option) or the medium model (-mm option) of the compiler is used, or the type of the memory model is not changed. (see NOTE)

(2) A pointer to a far array is used as the type name in an operand for the sizeof operator.

NOTE:

If the type of the memory model is not changed, the problem arises because the medium model is selected by default.

Example for condition 2:

```
----------------------------------------------------------------
[*.C]
int i3;

void func(void)
{
. . . . . .
   i3 = sizeof(__far int (*)[10]);  // (2)
. . . . . .
}
----------------------------------------------------------------
```

In the example above, the size returned is that of (int (*)[10]) because the far qualifier is not effective. Thus, it is interpreted as (__near int (*)[10]) in the small and medium models and, as a result, the size is output by mistake.

Example of code output under condition 2:

```
----------------------------------------------------------------
; line    22 :     i3 = sizeof(__far int (*)[10]);
;       Incorrect code              Correct code
        movw    ax,#02H ; 2         ; movw ax,#04H ; 4
        movw    !_i3,ax             : movw !_i3,ax
----------------------------------------------------------------
```

Condition 3:

Incorrect code is output when both conditions below are met:

(1) The large model (-ml option) of the compiler is used.

(2) A pointer to a near array is used as the type name in an operand for the sizeof operator.

Example for condition 3:

```
-----------------------------------------------------------------
[*.C]
int i4;

void func(void)
{
. . . . . .
   i4 = sizeof(__near int (*)[10]);  // (2)
. . . . . .
}
-----------------------------------------------------------------
```

In the example above, the size returned is that of (int (*)[10]) because the near qualifier is not effective. Thus, it is interpreted as (__far int (*)[10]) in the large model, and, as a result, the size is output by mistake.

Example of code output under condition 3:

```
-----------------------------------------------------------------
; line    23 :     i4 = sizeof(__near int (*)[10]);
;       Incorrect code            Correct code
        movw ax,#04H   ; 4         ; movw ax,#02H   ; 2
        mov  ES,#highw (_i4)      ; mov  ES,#highw (_i4)
        movw ES:!_i4,ax            ; movw ES:!_i4,ax
-----------------------------------------------------------------
```

2.4 Workarounds

Use a typedef to describe the type name when defining a type of the array pointer for processing of the near or far qualifier.

Workaround for condition 1:

```
-----------------------------------------------------------------
typedef __far int (*PA)[10];

void func(void)
{
. . . . . .
   pa2 = (PA)p2;   // (a)
   i1 = (*(PA)pa1)[0];  // (b)
. . . . . .
}
-----------------------------------------------------------------
```

Workaround for condition 2:

```
------------------------------------------------------------------
typedef __far int (*PA)[10];

void func(void)
{
. . . . . .
   i3 = sizeof(PA);
. . . . . .
}
------------------------------------------------------------------
```

Workaround for condition 3:

```
------------------------------------------------------------------
typedef __near int (*PA)[10];

void func(void)
{
. . . . . .
    i4 = sizeof(PA);
. . . . . .
}
------------------------------------------------------------------
```

3. Problem with Incorrect Code Being Output from Multiplication, Division, Remainder Arithmetic, and Indirect Reference Expressions

3.1 Products and Versions Concerned

CA78K0R V1.20 to V1.60
(included in the integrated development environment CubeSuite+)
CA78K0R V1.00 to V1.10
(included in the integrated development environment CubeSuite)
CC78K0R V1.00 to V2.13
(bundled with the integrated development environment PM+)

3.2 Description

In some cases, the output code may be incorrect if the operand of a multiplication, division, or remainder arithmetic operation in the char, signed char or unsigned char type includes an array element whose subscript is not a constant or an indirect reference expression using a pointer.

3.3 Conditions

This problem arises if the following conditions are all met:
(1) -qc optimization is made effective by a compiler option.
Any of the following actions makes -qc effective.
- Using the -qc option

- Using the -qx (x = 1, 2, 3) option
- Using the -q option by omitting the optimization type
- Omitting the -q option

(2) There is a binary operation in the char, signed char, or unsigned char type.

(3) The operand on the left of item (2) above is the result of an operation on an operand of the char, signed char, or unsigned char type.

(4) The operand on the right of the expression described by (2) above is the result of a multiplication, division, or remainder arithmetic operation on an operand of the char, signed char, or unsigned char type.

(5) The operand on the left or right of the expression described by (4) above includes an array element expression whose subscript is not a constant or an indirect reference expression using a pointer.

3.4 Example

```
------------------------------------------------------------------
[*.C]
unsigned char x1, uca1[5], uc1, uc2;
unsigned int ui1;
void func(void)
{
  x1 = (uc1 + 1) & (uca1[ui1] * uc2);   // (2),(3),(4) and (5)
}
------------------------------------------------------------------
```

Example of code output:

```
------------------------------------------------------------------
; line      5 :    x1 = (uc1 + 1) & (uca1[ui1] * uc2);
  mov    a,!_uc1
  inc      a                ; The results of the (uc1 + 1) operation
                             ; remain in the A register.
  movw   bc,!_ui1
  mov     a,_uca1[bc]     ; The register is overwritten.
  mov     x,!_uc2
  mulu    x
  mov     a,b
  and     a,x
  mov     !_x1,a
------------------------------------------------------------------
```

3.5 Workaround

Provide a temporary variable to perform the operation while substituting the results of the operation on one-byte data to a temporary variable.

Example:

```
------------------------------------------------------------
unsigned char x1, uca1[5], uc1, uc2;
unsigned int ui1;
void func()
{
   unsigned char temp; // temp variable provided.
   temp = uc1 + 1;    // Operation results on one-byte data
                      assigned to temp variable.
   x1 = temp & (uca1[ui1] * uc2); // Operate temp variable and
                                  indirect reference expression.
}
------------------------------------------------------------
```

4. Problem with the Assert Function Not Operating Normally

4.1 Products and Versions Concerned

CA78K0R V1.20 to V1.60
(included in the integrated development environment CubeSuite+)
CA78K0R V1.00 to V1.10
(included in the integrated development environment CubeSuite)
CC78K0R V1.00 to V2.13
(bundled with the integrated development environment PM+)

4.2 Description

When the small model or medium model is selected as the type of the memory model, or the type of the memory model is not changed (see NOTE), the assert function does not operate normally.

NOTE:
If the type of the memory model is not changed, the problem arises because the medium model is selected by default.

4.3 Workaround

Use the assert_f function instead of the assert function.

5. Problem with Return of an Error When a One-Bit-Wide Bit Field is Used in a Conditional Expression

5.1 Products and Versions Concerned

CA78K0R V1.50 to V1.60
(included in the integrated development environment CubeSuite+)
(see NOTE)

NOTE:
For the version of CubeSuite+ V1.03.00 or later.

5.2 Description

In some cases, the error "C0101: Internal error" may be returned when a one-bit-wide bit field is used in a conditional expression.

5.3 Conditions

This problem arises if the following conditions are all met:

(1) The following operator is used for a conditional expression.

- Comparison (equality and inequality) operators:
  ==, !=, <, >, <=, and >=

(2) The operand on the left of the expression described by (1) above is a constant and either 0 or 1.

(3) The operand on the right of the expression described by (1) is a one-bit-wide bit field having a constant even address in the range from 0FFE20H to 0FFFFFH.

5.4 Example

```
------------------------------------------------------------------
struct _st {c
   unsigned int b0:1;
   unsigned int b1:1;
   unsigned int b2:1;
};
#define bitsfr(n)   (((struct _st *)0xffd0)->b ## n) // (3) Bit field
                                                     // in 0ffd0H
int i;

void func(void)
{
   if (0 == bitsfr(1)) i++; // (1) and (2)
}
------------------------------------------------------------------
```

5.5 Workaround

Rearrange the conditional expression so that the constant is on the right.

Example:

```
------------------------------------------------------------------
struct _st {
   unsigned int b0:1;
   unsigned int b1:1;
   unsigned int b2:1;
};
#define bitsfr(n)   (((struct _st *)0xffd0)->b ## n)
```

```
int i;

void func(void)
{
   if (bitsfr(1) == 0) i++;   // Constant rearranged on the right.
}
```

---

6. Problem with the Conversion of Character Strings by the Strtol and Strtoul Functions Producing Incorrect Numerical Values

6.1 Products and Versions Concerned

CA78K0R V1.20 to V1.60
(included in the integrated development environment CubeSuite+)
CA78K0R V1.00 to V1.10
(included in the integrated development environment CubeSuite)
CC78K0R V1.00 to V2.13
(bundled with the integrated development environment PM+)

6.2 Description

When character strings are converted into numerical values by the strtol and strtoul functions, in some cases, the value may overflow and be returned as conversion has not been performed correctly.

6.3 Conditions

This problem arises if the following conditions are all met:
(1) Character strings are converted into numerical values by the strtol and strtoul functions.
(2) In the processing of functions described in (1) above, conversion to numerical values is performed from the head of a character and a carry for each 0x10000 is produced during conversion.
A carry for each 0x10000 is produced when the conditions below is met while conversion is being performed.

((N * base) / 0x10000) != ((N * base + c) / 0x10000)

N: The value obtained by converting a character string from its head to the n-th character
base: The radix
c: The value of the n + 1-th character

Example:
When strtol ("65537", &err, 10) is executed, 6553 (the value obtained by converting up to the 4th character), 10 (the radix), and 7 (the 5th character) are obtained.

Left side = ((6553 * 10) / 0x10000) = 0
Right side = ((6553 * 10 + 7) / 0x10000) = 1

Left side != right side is true, and the value returned has overflowed.

6.4 Workaround

When the radix is 10 in the strtol function, replace it with the atol function.
When the radix is 10 in the strtoul function and the value obtained by the conversion can be expressed by a signed long integer, replace it with the atol function.
If the method above does not work, there is no workaround.

7. Problem with Incorrect Address Reference Being Made in the Case of Reference to a Symbol Resolved by the Assembler for the RL78-S1 Core

7.1 Products and Versions Concerned

CA78K0R V1.50 to V1.60
(included in the integrated development environment CubeSuite+)
(see NOTE)

NOTE:
For the version of CubeSuite+ V1.03.00 or later.

7.2 Description

For the RL78-S1 core, incorrect code is returned for reference to a symbol belonging to a segment specified as an absolute address by either of the following:

- Directives having "#word", "ES:!addr16", "ES:word[B]", "ES:!word[C]" or "ES:word[BC]" as an operand
- DW or DG directive

7.3 Conditions

This problem arises if the following conditions are all met:
(1) A microcontroller with the RL78-S1 core is in use.
(2) Reference is made to a symbol that belongs to a segment specified by an absolute address.
(3) The symbol reference format described in (2) above is any of those listed below.
(a) ES:!addr16
(b) ES:word[B]
(c) ES:word[C]
(d) ES:word[BC]
(e) #word
(f) DW word
(g) DG lword

7.4 Example

[Sample.ASM] (Assembler source file)

```
------------------------------------------------------------------------
C1  CSEG   AT  30H       ; Condition (2)
TAB1:
   DB   1
   DB   2

   CSEG
main:
   MOV      A,ES:!TAB1       ; Condition (3)(a)
   MOV      A,ES:TAB1[B]     ; Condition (3)(b)
   MOV      A,ES:TAB1[C]     ; Condition (3)(c)
   MOV      A,ES:TAB1[BC]    ; Condition (3)(d)
   MOVW     AX,#TAB1         ; Condition (3)(e)
   MOVW     AX,ES:!TAB1      ; Condition (3)(a)
   MOVW     AX,ES:TAB1[B]    ; Condition (3)(b)
   MOVW     AX,ES:TAB1[C]    ; Condition (3)(c)
   MOVW     AX,ES:TAB1[BC]   ; Condition (3)(d)
   RET
;
   DW    TAB1          ; Condition (3)(f)
   DG    TAB1          ; Condition (3)(g)
   END
------------------------------------------------------------------------
```

[Sample.P] (Assemble list file) Condition

```
------------------------------------------------------------------------------
-----------------------
ALNO  STNO  ADRS   OBJECT     M I      SOURCE STATEMENT
1     1     -----  C1         CSEG     AT      30H       (2)
2     2     00030  TAB1:
3     3     00030  01         DB       1
4     4     00031  02         DB       2
5     5
6     6     -----  CSEG
7     7     000CE  main:
8     8     000CE  118F3080   MOV      A,ES:!TAB1        (3)(a)
9     9     000D2  11093080   MOV      A,ES:TAB1[B]      (3)(b)
10    10    000D6  11293080   MOV      A,ES:TAB1[C]      (3)(c)
11    11    000DA  11493080   MOV      A,ES:TAB1[BC]     (3)(d)
12    12    000DE  303080     MOVW     AX,#TAB1          (3)(e)
13    13    000E1  11AF3080   MOVW     AX,ES:!TAB1       (3)(a)
14    14    000E5  11593080   MOVW     AX,ES:TAB1[B]     (3)(b)
15    15    000E9  11793080   MOVW     AX,ES:TAB1[C]     (3)(c)
16    16    000ED  11793080   MOVW     AX,ES:TAB1[BC]    (3)(d)
17    17    000F1  D7         RET
18    18    ;
```

```
19      19      000F2   3080        DW      TAB1              (3)(f)
20      20      000F4   30800F00    DG      TAB1              (3)(g)
21      21      END
-------------------------------------------------------------------------------
-----------------------
```

7.5 Workaround

Comment out the specification of the segment as an absolute address. Instead, specify the area where it is allocated in the link directive file.

[Sample.ASM] (Assembler source file)

```
-------------------------------------------------------------------------
C1  CSEG    ;AT 30H     ; Comment out the specification of the segment
                            ; as an absolute address.
TAB1:
   DB    1
   DB    2

   CSEG
main:
   MOV      A,ES:!TAB1       ; Condition (3)(a)
   MOV      A,ES:TAB1[B]     ; Condition (3)(b)
   MOV      A,ES:TAB1[C]     ; Condition (3)(c)
   MOV      A,ES:TAB1[BC]    ; Condition (3)(d)
   MOVW     AX,#TAB1         ; Condition (3)(e)
   MOVW     AX,ES:!TAB1      ; Condition (3)(a)
   MOVW     AX,ES:TAB1[B]    ; Condition (3)(b)
   MOVW     AX,ES:TAB1[C]    ; Condition (3)(c)
   MOVW     AX,ES:TAB1[BC]   ; Condition (3)(d)
   RET
;
   DW    TAB1           ; Condition (3)(f)
   DG    TAB1           ; Condition (3)(g)
   END
-------------------------------------------------------------------------
```

[Sample.DR] (Link directive file)

```
-------------------------------------------------------------------------
MERGE C1 : AT (30H)     ; Use the link directive file to specify the
                            ; area where it is allocated.
-------------------------------------------------------------------------
```

[Sample.P] (Assemble list file) Condition

```
------------------------------------------------------------------------------
-----------------------
ALNO  STNO  ADRS   OBJECT      M I     SOURCE STATEMENT
1     1     -----  C1          CSEG    AT      30H       (2)
2     2     00030  TAB1:
3     3     00030  01          DB      1
4     4     00031  02          DB      2
5     5
6     6     -----  CSEG
7     7     000CE  main:
8     8     000CE  R118F3000   MOV     A,ES:!TAB1        (3)(a)
9     9     000D2  R11093000   MOV     A,ES:TAB1[B]      (3)(b)
10    10    000D6  R11293000   MOV     A,ES:TAB1[C]      (3)(c)
11    11    000DA  R11493000   MOV     A,ES:TAB1[BC]     (3)(d)
12    12    000DE  R303000     MOVW    AX,#TAB1          (3)(e)
13    13    000E1  R11AF3000   MOVW    AX,ES:!TAB1       (3)(a)
14    14    000E5  R11593000   MOVW    AX,ES:TAB1[B]     (3)(b)
15    15    000E9  R11693000   MOVW    AX,ES:TAB1[C]     (3)(c)
16    16    000ED  R11793000   MOVW    AX,ES:TAB1[BC]    (3)(d)
17    17    000F1  D7          RET
18    18    ;
19    19    000F2  R3000       DW      TAB1              (3)(f)
20    20    000F4  R30000000   DG      TAB1              (3)(g)
21    21    END
------------------------------------------------------------------------------
-----------------------
```

8. Problem with Incorrect Code Being Output for the CALL Directive

8.1 Products and Versions Concerned

CA78K0R V1.20 to V1.60
(included in the integrated development environment CubeSuite+)
CA78K0R V1.00 to V1.10
(included in the integrated development environment CubeSuite)
RA78K0R V1.31 to V1.80
(bundled with the integrated development environment PM+)

8.2 Description

When the CALL directive is used, even if the displacement to the branch destination is less than -8000H or more than +7FFFH, branching will not be correct as the incorrect code generated is for CALL $!addr20. (see NOTE)
This also becomes applicable to C source code if the CALL directive is used in an ASM statement.

NOTE:
The CALL $!addr20 is 3bytes directive.

8.3 Conditions

This problem arises if the following conditions are all met:

(1) Assembler code includes a CALL directive or an ASM statement in C source code contains a CALL directive.

(2) The CALL directive and symbol for the branch destination are in the same file and belong to the same absolute segment or the same segment with the BASE relocation attribute.

(3) The CALL or BR directive is in a lower address in the same segment as the symbol for the branch destination. Also, in the CALL or BR directive, if either of the following is met:

- The address where the branch destination is allocated when the size of CALL or BR directive in the lower-order address becomes minimum is in the range from 0H to 0FFFFH.
- The address where the branch destination is allocated when the size of CALL or BR directive in the lower-order address becomes maximum is out of the range from 0H to 0FFFFH.

(4) Displacement from the CALL directive to the branch destination is less than -8000H or more than +7FFFH.

8.4 Workaround

Write the relevant CALL directive as a CALL directive with immediate addressing in the way shown below.

Example:

```
-----------------------------------------------------
. . . . . .
   CALL   !!addr20
. . . . . .
-----------------------------------------------------
```

Give the label for the branch destination as addr20.

9. Problem with the BR and CALL Directives Producing Errors

9.1 Products and Versions Concerned

CA78K0R V1.20 to V1.60
(included in the integrated development environment CubeSuite+)
CA78K0R V1.00 to V1.10
(included in the integrated development environment CubeSuite)
RA78K0R V1.31 to V1.80
(bundled with the integrated development environment PM+)

9.2 Description

There are cases where the error "E2410: Phase error" is returned when a BR or CALL directive is used. This also applies to C source code where an ASM statement contains a BR or CALL directive.

9.3 Conditions

This problem arises if the following conditions are all met:

(1) Assembler code includes a BR or CALL directive or an ASM statement in C source code contains a BR or CALL directive.

(2) Either the BR directive that makes a forward reference to the label for the branch destination or the CALL directive and branch destination are in a single absolute segment. Displacement to the branch destination is at least 80H.

(3) The absolute segment in (2) above also contains another BR directive or CALL directive. Also, the branch destination of either of these directives is in an absolute segment other than (2) above and any of conditions from (a) to (c) below apply.

(a) The branch destination of the BR or CALL directive is near the address 10000H. Near the address 10000H shows the following point:

- The point where BR/CALL !addr16 switches to BR/CALL !!addr20
- The point where BR/CALL !addr16 switches to BR/CALL $!addr20

(b) Displacement to the branch destination is near 80H for the BR directive. Near 80H shows the following point:

- The point where BR $addr20 switches to BR $!addr20
- The point where BR $addr20 switches to BR !addr16

(c) Displacement to the branch destination is near 8000H for the BR or CALL directive. Near 8000H shows the following point:

- The point where BR/CALL $!addr20 switches to BR/CALL !!addr20

(4) A BR or CALL directive in an absolute segment other than (2) above is allocated to an address lower than the branch destination label of (3) above.

9.4 Example

E2410: Phase error is returned for the line with the label L1 if the code is as shown below.

[Sample.ASM] (Assembler source file)

```
------------------------------------------------------------------------
        ORG     8000H   ; Absolute segment A
        BR      L1          ; BR directive under condition (2)
        DS      (80H)
L1:                         ; Branch destination under condition (2) (label)
        CALL    L2          ; CALL directive under condition (3)

        ORG     10000H  ; Absolute segment B
        DS      (82H)
        BR      L3          ; BR directive under condition (4)
L2:                         ; Branch destination under condition (3) (c)
```

```
        NOP
    L3:
    ------------------------------------------------------------------------
```

9.5 Workaround

Comment out the absolute address specification of the segment. Instead, use the link directive file to specify the allocation.

Example:

[Sample.ASM] (Assembler source file)

```
    ------------------------------------------------------------------------
    A1      CSEG              ; Relocatable segment A1
            ;ORG    8000H     ; Comment out the absolute address specification
                              ; of segment A.
            BR      L1        ; BR directive under condition (2)
            DS      (80H)
    L1:                       ; Branch destination under condition (2) (label)
            CALL    L2        ; CALL directive under condition (3)

    B1      CSEG              ; Relocatable segment B1
            ;ORG    10000H    ; Comment out the absolute address specification
                              ; of segment B
            DS      (82H)
            BR      L3        ; BR directive under condition (4)
    L2:                       ; Branch destination under condition (3) (c)
            NOP
    L3:
    ------------------------------------------------------------------------
```

[Sample.DR] (Link directive file)

```
    ------------------------------------------------------------------------
    MERGE   A1 : AT (8000H)  ; Specify the allocation of relocatable
                             ; segment A1.
    MERGE   B1 : AT (10000H) ; Specify the allocation of relocatable
                             ; segment B1.
    ------------------------------------------------------------------------
```

10. Schedule for Fixing the Problem

These problems will be fixed in CubeSuite+ CA78K0R compiler V1.70 (to be published on January 30, 2014).
For CubeSuite (see NOTE) CA78K0R compiler, PM+ CC78K0R compiler and PM+ RA78K0R assembler, we have no plan to fix this problem.

NOTE:
CubeSuite is a discontinued product.

[Disclaimer]
The past news contents have been based on information at the time of publication. Now changed or invalid information may be included. The URLs in the Tool News also may be subject to change or become invalid without prior notice.