Microcontroller Technical Information

78K0R C Compiler CC78K0R Usage Restrictions	Document No.	ZMT-F35-11-0002	1/2
	Date issued	April 7, 2011	
	Issued by	MCU Tool Product Marketing Department MCU Software Division MCU Business Unit Renesas Electronics Corporation	
Related documents CC78K0R Ver. 2.00 Language: U18548EJ1V0UM00 CC78K0R Ver. 2.00 Operation: U18549EJ1V0UM00 78K0R C Compiler CC78K0R Ver. 2.13 Operating Precautions: ZUD-CD-10-0100	Notification classification	√	Usage restriction
			Upgrade
			Document modification
			Other notification

1. Affected product

   CC78K0R (For the affected versions, see *1. Product History* on Attachment 1.

2. New restrictions

   The following restrictions have been added. For details, see the attachment.

   - No. 30 The C0101 error occurs when the `-qj` option is specified.
   - No. 31 Restriction on type-casting addresses in the `far` area to a `long` or `unsigned long`.
   - No. 32 Macro expansion using the `##` operator results in an error.
   - No. 33 Symbol information for an interrupt function is not output to the assembler source.
   - No. 34 Code that specifies the ES register settings might not be output.

3. Workarounds

   The following workarounds are available for this restriction. For details, see the attachment.

   - No. 30 Do either of the following:
     1. Disable the `-qj` option.
     2. Delete the code that is not executed or embed the code between `#if 0` and `#endif` so as not to make it subject to compiling.
   - No. 31 Do not type-cast an address to a `long` or `unsigned long` if the initial value is used for the address.
   - No. 32 Do either of the following:
     1. Do not use the `##` operator.
     2. Place a function-like macro parameter immediately after the `##` operator.
   - No. 33 Define the interrupt function or output the object module file.
   - No. 34 Immediately after incrementing or decrementing a floating point variable, insert a dummy code that references the variable by using a pointer.

4. Modification schedule

   Restrictions No. 30 to No. 34 will be removed in the next revision.

5. List of restrictions

A list of restrictions in the CC78K0R, including the revision history and detailed information, is described on the attachment.

6. Revision history

CC78K0R 78K0R C Compiler Usage Restrictions

Document Number	Date Issued	Description
ZBG-CD-06-0088	December 21, 2006	First edition
ZBG-CD-07-0012	January 29, 2007	Addition of restriction No. 20
ZBG-CD-07-0082	November 29, 2007	Addition of restrictions No. 21 and No. 22
ZBG-CD-08-0040	September 18, 2008	Addition of restrictions No. 23 to No. 26
ZBG-CD-09-0026	May 20, 2009	Addition of restrictions No. 27 and No. 28
ZBG-CD-10-0014	March 30, 2010	Addition of restriction No. 29
ZMT-F35-11-0002	April 7, 2011	Addition of restrictions No. 30 to No. 34

List of Restrictions in CC78K0R

1. Product History

No.	Bugs and Changes/Addition to Specifications	Version						
		1.00	1.10	1.20	2.00	2.10	2.12	2.13
1	The `norec` function cannot be used.	×	–	–	–	–	–	–
2	Extended functions for a real-time OS cannot be used.	×	○	○	○	○	○	○
3	The compact memory model cannot be used.	×	–	–	–	–	–	–
4	A function whose name length is 249 characters is called as a function that has a 248-character name.	×	○	○	○	○	○	○
5	Function information in assembler source becomes invalid.	×	○	○	○	○	○	○
6	When defining the return value of a function, if a type qualifier is used to the right of a type specifier and the type specifier is `struct`, `union`, `enum`, or a `typedef` name, the type specifier is output as an `int`.	×	○	○	○	○	○	○
7	An error occurs during assembly if no functions are defined in a C source file in which extended interrupt functions (`#pragma vect` and `#pragma interrupt`) are used.	×	○	○	○	○	○	○
8	Initializing an external variable declared as `extern` in a block does not cause an error. In addition, the debugging information output to the assembler source is incorrect.	×	×	×	○	○	○	○
9	Linking a variable that has the same name to a variable declared as `extern` in a block might be invalid.	×	×	×	×	○	○	○
10	A multidimensional array that has an undefined size might not work properly.	×	×	×	×	○	○	○
11	When initializing an array whose size is not defined, if elements in the initializer braces are enclosed inconsistently, an area of invalid size is allocated.	×	×	×	×	○	○	○
12	The warning message W0411 is output when assigning an array address to a pointer if the structure of the array pointed to by the pointer is the same as that of the array whose address is referenced.	×	○	○	○	○	○	○
13	Optimizing a branch instruction might result in the error C0101 when outputting assembler source in the medium or large model.	×	○	○	○	○	○	○
14	Invalid code might be output if a `directmap` variable is allocated to an area extending over the boundary of areas for which short-direct and SFR addressing are possible.	×	○	○	○	○	○	○
15	The error E0301 is output if an unlabeled array parameter using a `typedef` name is declared.	×	○	○	○	○	○	○
16	Invalid code is output for unconditional branch instructions.	×	○	○	○	○	○	○
17	When the data type of the `volatile` type qualifier is converted, the `volatile` type qualifier code might not be output.	×	×	○	○	○	○	○
18	The display in the debugger might be invalid if the `-qj` option is not specified.	×	○	○	○	○	○	○

×: Applicable, ○: Not applicable, –: Not relevant, *: Check tool available

No.	Bugs and Changes/Addition to Specifications	Version						
		1.00	1.10	1.20	2.00	2.10	2.12	2.13
19	No error is output if a character string constant is specified for the address at which a `char` (either signed or unsigned) array-type `directmap` variable is to be allocated to the `far` area.	×	×	×	○	○	○	○
20	Invalid code is output if `typedef`, `struct`, `union`, or `enum` is declared in a function.	×*	×*	○	○	○	○	○
21	Invalid code is output when performing indirect reference if the `-ra` or `-rc` option is specified.	×	×	×*	○	○	○	○
22	The ES setting code stored in an array or pointer might not be output.	×	×	×*	○	○	○	○
23	Invalid code is output if the last element of an initializer list for a `char`, `signed char`, or `unsigned char` array is a character string and one or more constants or character constants are placed before the character string.	×	×	×*	×*	○	○	○
24	Invalid code is output if referencing a pointer that has an offset obtained by subtracting one pointer from another.	×	×	×*	×*	○	○	○
25	Invalid code is output if the `-qc` option has not been specified (sign extension is specified to be `int` type).	×	×	×*	×*	○	○	○
26	Invalid code is output as a result of the BCD operation function `adbcdw` or `sbbcdw`.	×	×	×	×	○	○	○
27	An error occurs if the `-ng` option is specified and a branch instruction is used in a function that includes the `asm` statement.	×	×	×	×	×	○	○
28	The line number is not output for a statement that immediately follows a nested `if` statement and is outside that statement's conditional block.	–	×	×	×	×	○	○
29	Restriction due to an instruction (`mov`, `movw`) that uses `word[BC]` as an operand accessing the incorrect address.	×	×	×*	×*	×*	×*	○
30	The C0101 error occurs when the `-qj` option is specified	×	×	×	×	×	×	×
31	Restriction on type-casting addresses in the `far` area to a `long` or `unsigned long`	–	–	–	–	×*	×*	×*
32	Macro expansion using the `##` operator results in an error	×	×	×	×	×	×	×
33	Symbol information for an interrupt function is not output to the assembler source	×	×	×	×	×	×	×
34	Code that specifies the ES register settings might not be output	×	×	×*	×*	×*	×*	×*

×: Applicable, ○: Not applicable, –: Not relevant, *: Check tool available

2. Details of Usage Restrictions

No. 1 The `norec` function cannot be used.

Description:

The `norec` function cannot be used. If used, the warning message W0332 is output and the `norec` function is handled as an ordinary function. This also applies if `__leaf` is used instead of `norec`.

Workaround:

There is no workaround.

Correction:

This will not be corrected, so regard it as a specification.

No. 2 Extended functions for a real-time OS cannot be used.

Description:

The following extended functions for a real-time OS cannot be used:

(1) Interrupt handler for RTOS (`#pragma rtos_interrupt...`)

(2) Interrupt handler qualifier for RTOS (`__rtos_interrupt`)

(3) Task function for RTOS (`#pragma rtos_task`)

Workaround:

There is no workaround.

Correction:

This issue has been corrected in V1.10.

No. 3 The compact memory model cannot be used.

Description:

The compact memory model cannot be used, nor can it be specified using the `-mc` option.

(1) When using PM+

The **Compact** option cannot be selected on the **Memory Model** tab in the **Compiler Options** dialog box, which is displayed by selecting **Compiler Options** in the **Tool** menu.

(2) When using the command line version

`-mc` cannot be used in the CC78K0R options.

Workaround:

Use the small model (by specifying the `-ms` option), and use the `__far` type qualifier to allocate variables that cannot be allocated in the small model.

Correction:

This will not be corrected, so regard it as a specification.

No. 4 A function whose name length is 249 characters is called as a function that has a 248-character name.

Description:

When using the medium memory model, large memory model, or `__far` type qualifier, if a function whose name length is 249 characters is called, the function is called as a function that has a 248-character name in the output assembler source, resulting in invalid code.

This restriction applies in the following two cases:

(1) If there is no function that has a 248-character name
An error occurs during linking.
(2) If there is another function that has a 248-character name
No error occurs but the function that has a 248-character name is called.

Example:

```
void
sub_func0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_0123456
789_0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_
0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_0123
456789(void)
{
                              ...
}
void main(void)
{
sub_func0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_0123456
789_0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_
0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_0123456789_0123
456789();
}
```

Workaround:
Do not use function names longer than 248 characters.
Correction:
This issue has been corrected in V1.10.

No. 5 Function information in assembler source becomes invalid.
Description:
When the large model is specified (by using the `-ml` option), the displayed function information (return values and parameters of defined functions) in assembler source becomes invalid, but only for functions that return a structure.

Example:

```
[test.c]
struct st {
       long l[2];
} gst;
struct st stfunc() {
       return gst;
}
void main() {
       gst = stfunc();
}
```

```
[test.asm]
     ...
       movw    bc,#loww (L0003) ★ Code is correct
       mov     e,#highw (L0003) ★ Code is correct
       ret
     ...
; *** Code Information ***
;
; $FILE D:\work\test.c
; $FUNC stfunc(4)
;      bc=(void)     ★ Incorrect display. bc,de is supposed to be displayed.
;      CODE SIZE= 24 bytes, CLOCK_SIZE= 21 clocks, STACK_SIZE= 0 bytes
;
; $FUNC main(7)
;      void=(void)
;      CODE SIZE= 32 bytes, CLOCK_SIZE= 31 clocks, STACK_SIZE= 4 bytes
;
; $CALL stfunc(8)
;      bc,de=(void)  ★ The calling side is displayed correctly.
```

Workaround:

There is no workaround.

Correction:

This issue has been corrected in V1.10.

No. 6 When defining the return value of a function, if a type qualifier is used to the right of a type specifier and the type specifier is `struct`, `union`, `enum`, or a `typedef` name, the type specifier is output as an `int`.

Description:

When defining the return value of a function, if a type qualifier is used to the right of a type specifier and the type specifier is `struct`, `union`, `enum`, or a `typedef` name, the type specifier is output as an `int`.

Debugging information on the defined function name becomes invalid.

Declaring a prototype that has the above sort of return value in the same file causes an E0747 error.

Declaring a prototype that has the above sort of return value in another file causes an E3403 error during linking.

Example:

```
struct t1 {
       char a[4];
       int b;
};
struct t1 __far *stp1;
struct t1 __far st1;

/* Prototype not declared */
struct t1 __far *func1()
```

```
{
       return stp1; /* W0411 occurs */
}

/* Prototype declared */
struct t1 __far func2();
struct t1 __far func2() /* E0747 occurs */
{
       return st1; /* E0402 occurs */
}
```

Example of invalid code:

```
typedef unsigned long DWORD;
unsigned long ldata1 = 0x12348765;
unsigned long ldata2;
DWORD __far func1()
{
       return ldata1;
}
void func()
{
       ldata2 = func1();
}
```

The value returned by function `func1()` is handled as an `int`, not an `unsigned long`, so the value of the variable `ldata2` becomes 0xffff8765, not 0x12348765.

Workaround 1:

Use a `typedef` name that includes the type specifier and type qualifier.

```
typedef struct t1 __far STR1;
STR1 *func1()
{
     return stp1;
}

STR1 func2();
STR1 func2()
{
     return st1;
}
```

Workaround 2:

Change the position of the type qualifier.

```
/* Prototype not declared */
__far struct t1 *func1()
{
     return stp1;
}

/* Prototype declared */
```

```
__far struct t1 func2();
__far struct t1 func2()
{
     return st1;
}
```

Correction:

This issue has been corrected in V1.10.

No. 7 An error occurs during assembly if no functions are defined in a C source file in which extended interrupt functions (`#pragma vect` and `#pragma interrupt`) are used.

Description:

If no functions are defined in a C source file in which extended interrupt functions (`#pragma vect` and `#pragma interrupt`) are used, outputting debugging information is specified (by using the `-g` option), and outputting the assembler source module file is specified (by using the `-a` or `-sa` option), the vector table and C source comments for the file are not output. An error occurs during assembly. These items are output normally to the object module file.

Example:

```
[vect.c]
#pragma interrupt INTP0 func
/* func() is defined in another module */
int dmy;

[vect_stub.c]
__interrupt void func(void) {}
void main(void) {}
```

Error message output during assembly:

```
vect.asm(42) : RA78K0R error E2404: Public symbol is undefined '_@vect08'
```

Workaround:

Do either of the following:

- Define at least one function in a C source file in which `#pragma vect` (or `interrupt`) is used.
- Do not output an assembler source module file (by using the `-a` or `-sa` option).

Example:

```
#pragma interrupt INTP0 func
/* func()is defined in another module */
void dmy_func() {}
```

Correction:

This issue has been corrected in V1.10.

No. 8 Initializing an external variable declared as `extern` in a block does not cause an error. In addition, the debugging information output to the assembler source is incorrect.

Description:

Because initializing an external variable declared as `extern` in a block is not compliant with the ANSI C specifications, it should cause an error, but the code does not. The compiler interprets the code as defining an external variable that has an initial value and outputs the code.

The compiler outputs the correct debugging information to the object file, but the debugging information output to the assembler source is incorrect.

Example:

```
int  i;
void  f(void) {
     extern int  i = 2;
}
```

Workaround:

There is no workaround.

Correction:

This issue has been corrected in V2.00.

No. 9 Linking a variable that has the same name to a variable declared as `extern` in a block might be invalid.

Description:

Linking a variable that has the same name to a variable declared as `extern` in a block is invalid in any of the following cases:

(1) If a variable declared as `extern` in a block and a variable declared as `static` outside that block or the subsequent blocks have the same name

No error occurs and linking is not performed, so, if this variable is referenced, invalid code is output.

Example:

```
void  f(void) {
      extern int  i;
      i = 1;                  /* Invalid code is output */
}
static int  i;
```

(2) If a variable declared as `extern` in a block and a variable not declared as `static` outside that block or the subsequent blocks have the same name

Linking is not performed and invalid code is output.

Example:

```
 void  f(void) {
       extern int  i;
       i = 1;                  /* Invalid code is output */
 }
int  i;
```

(3) If a variable declared as `extern` in a block and a variable not declared as `extern` outside the block before the `extern` variable have the same name, and an automatic variable that has the same name is declared in the block containing the `extern` variable
The variable outside the block and the variable declared as `extern` in the block are not linked, and invalid code is output.
Example:

```
int i = 1;
void  f(void) {
      int  i;
      {
               extern int  i;
               i = 1;         /* Invalid code is output */
      }
}
```

(4) If a variable declared as `extern` in a block and a variable declared as `extern` in another block have the same name
Linking is not performed, and invalid code is output.
Example:

```
void  f1(void) {
      extern int  i;
      i = 2;
}
void  f2(void){
      extern int  i;
      i = 3;
}
```

Workaround:
There is no workaround.
Correction:
This issue has been corrected in V2.10.

No. 10 A multidimensional array that has an undefined size might not work properly.
Description:
A multidimensional array that has an undefined size might not work properly.
Example 1:

```
char  c[][3]={{1},2,3,4,5};        /* Invalid code */
```

Example 2:

```
char  c[][2][3]={"ab","cd","ef"};  /* Error (E0756) */
```

Workaround:
Define the size of the multidimensional array.
Correction:
This issue has been corrected in V2.10.

No. 11 When initializing an array whose size is not defined, if elements in the initializer braces are enclosed inconsistently, an area of invalid size is allocated.

Description:

When initializing an array whose size is not defined, if elements in the initializer braces are enclosed inconsistently, an area of invalid size is allocated.

Example:

```
struct t {
    int a;
    int b;
} x[] = {1, 2, {3, 4}};
```

Workaround:

Do either of the following:

(1) Make sure the enclosing braces are consistent.

```
struct t {
    int a;
    int b;
} x[] = {{1, 2}, {3, 4}};
```

(2) Define the size of the array.

```
struct t {
    int a;
    int b;
} x[2] = {1, 2, {3, 4}};
```

Correction:

This issue has been corrected in V2.10.

No. 12 The warning message W0411 is output when assigning an array address to a pointer if the structure of the array pointed to by the pointer is the same as that of the array whose address is referenced.

Description:

The warning message W0411 is output when assigning an array address to a pointer if the structure of the array pointed to by the pointer is the same as that of the array whose address is referenced.

Example:

```
char a1[2];
char a2[2][3];
char (*p1)[2];
char (*p2)[2][3];
char (*p3)[3];
void func()
{
       p1 = &a1;       /* Code is correct but W0411 is output */
       p2 = &a2;       /* Code is correct but W0411 is output */
       p3 = &a2[1];    /* Code is correct but W0411 is output */
}
```

Workaround:

Code as follows so as not to reference the array address:

Example:

```
char a1[2];
char a2[2][3];
char *p4;
char (*p5)[3];
void func()
{
       p4 = a1;
       p5 = a2;
}
```

Note: There is no workaround for the third error listed above.

Correction:

This issue has been corrected in V1.10.

No. 13 Optimizing a branch instruction might result in the error C0101 when outputting assembler source in the medium or large model.

Description:

Optimizing `br !!addr20` to `br $!addr20` might result in the error C0101 when outputting assembler source in the medium or large model.

If this error is not output, the object module file and assembler source file output by the CC78K0R are correct.

Workaround:

Do not output the assembler source file from the CC78K0R.

Correction:

This issue has been corrected in V1.10.

No. 14 Invalid code might be output if a `directmap` variable is allocated to an area extending over the boundary of areas for which short-direct and SFR addressing are possible.

Description:

Invalid code might be output if a `directmap` variable is allocated to an area extending over the boundary of areas for which short-direct and SFR addressing are possible. If such code is assembled, an error such as `Operand out of range (saddr)` or `Illegal operand` is output.

Example:

```
#define ADDR    0xfff1f
__directmap struct t1 {
    char c1;
    char c2;
} dst1 = { ADDR };
__directmap int di1 = ADDR;
void func()
{
    ++dst1.c2;
```

```
        di1 += 2;
    }


  [.asm]
    ; line        13 :        ++dst1.c2;
          inc           _dst1+1             ; invalid code
    ; line        14 :        di1 += 2;
          add       _di1,#02H
          addc          _di1+1,#00H         ; invalid code
```

Workaround:

Do not allocate a `directmap` variable to an area extending over the boundary of areas for which short-direct and SFR addressing are possible.

Correction:

This issue has been corrected in V1.10.

No. 15 The error E0301 is output if an unlabeled array parameter using a `typedef` name is declared.

Description:

The error E0301 is output if an unlabeled array parameter using a `typedef` name is declared.

Example:

```
typedef int TYP;
void func(TYP [4]); /* E0301: Syntax error */
```

Workaround:

Declare a labeled array parameter.

```
typedef int TYP;
void func(TYP a[4]);
```

Correction:

This issue has been corrected in V1.10.

No. 16 Invalid code is output for unconditional branch instructions.

Description:

Invalid code is output for unconditional branch instructions under the conditions below. The output assembler source is normal, however.

(1) `BR !addr16` instruction in a `callt` or interrupt function

The jump destination exceeds the range of –128 to 127 bytes from the address following the `BR` instruction

(2) `BR !!addr20` instruction in a `far` function

The jump destination exceeds the range of –32,768 to 32,767 bytes from the address following the `BR` instruction

Workaround:

Create an object module file by assembling the assembler source output from the CC78K0R, using the RA78K0R.

Correction:

This issue has been corrected in V1.10.

No. 17 When the data type of the `volatile` type qualifier is converted, the `volatile` type qualifier code might not be output.

Description:

When the data type of the `volatile` type qualifier is converted, the `volatile` type qualifier code might not be output.

If `adrs[id]` is handled as an SFR address as shown in the example below, the intended code is not output. In addition, the code does not specify word access.

Example:

```
volatile unsigned short *adrs[] = {
        0xff40,
        0xff42,
        0xff44,
 };
 int id;
 unsigned short s1;
 void func()
 {
        s1 = *adrs[id] & 0x8000;
 }
```

Workaround:

Cast the operands explicitly by using the `volatile` type qualifier as follows:

```
s1 = *(volatile unsigned int *)adrs[id] & 0x8000;
```

Correction:

This issue has been corrected in V1.20.

No. 18 The display in the debugger might be invalid if the `-qj` option is not specified.

Description:

The display in the debugger might be invalid if either of the following is satisfied when the `-qj` option is not specified:

(1) When the execution goes through the first compound statement (the true part) of an `if-else` statement in nested code and then exits the `if-else` statement, the execution goes through the second compound statement (the false part), which should not be executed.
This condition does not apply if the `else` statement is an empty statement.

(2) When the execution exits an `if` statement without going through a compound statement without an `else` statement (the true part) in nested code, the execution goes through the compound statement (the false part) that should not be executed.

Example 1:

```
int a=1, b=1, c;
void func1()
{
    if (a) {
        if (b) {
            c = 1;
        }
        else {
            c = 2;
        }
    }
    else {
        c = 3;
    }
}
```

Example 2:

```
int a=1, b=0, c=0;
void func2()
{
    if (a) {
        if (b) {
            c = 1;
        }
    }
    else {
        c = 2;
 }
}
```

Workaround:

Specify the `-qj` option.

Correction:

This issue has been corrected in V1.10.

No. 19 No error is output if a character string constant is specified for the address at which a `char` (either signed or unsigned) array-type `directmap` variable is to be allocated to the `far` area.

Description:

No error is output if a character string constant is specified for the address at which a `char` (either signed or unsigned) array-type `directmap` variable is to be allocated to the `far` area, but the code results in allocation at address 0.

If using the large model, this restriction applies even if no `__far` qualifier is used.

Example:

```
__directmap __far char cary[10] = {"string"};
char c;
void func(void) {
     c = cary[5];
}
```

Workaround:

Specify the allocation address using integers.

Correction:

This issue has been corrected in V2.00.

No. 20 Invalid code is output if `typedef`, `struct`, `union`, or `enum` is declared in a function.

Description:

Code that corrupts the HL register contents is output if `typedef`, `struct`, `union`, or `enum` is declared in a function.

Invalid code example:

Code that specifies a frame pointer is output to a function that does not need the frame (does not need to save and restore the HL register contents).

```
_func1:
    push    ax
    push    bc
    movw    hl,sp   ★ Corrupting code
```

Applicable examples:

This restriction applies to the functions `func1` to `func5`.

It also applies when no interrupt function (`_interrupt`) is specified.

Example 1:

```
__interrupt void func1() {
        typedef int T1;
}

__interrupt void func2() {
       struct S1 {
            int i;
        };
}

__interrupt void func3() {
       union U1 {
           int i;
        };
}

__interrupt void func4() {
       enum E1 {
           E11
        };
}
```

The restriction applies to `func1` to `func4`.

Example 2:

```
__interrupt void func5() {
        static union U2 {
            int i;
        } u2;
}
```

The restriction applies because the static variable `u2` in the function is not generated in the frame.

Invalid code is not output under the following conditions:

(1) The function is a `_flash` function, an RTOS interrupt function, or an RTOS task function. (RTOS interrupt functions and RTOS task functions are not supported in CC78K0R V1.00.)

(2) The `-qr` option is not specified (which is the default) and either of the following conditions is satisfied:

2-1. The problematic function is an interrupt function that includes an automatic variable or function call.

2-2. The problematic function is an ordinary function that includes a parameter or automatic variable.

(3) The `-qr` option is specified (which is not the default) and either of the following conditions is satisfied:

3-1. The problematic function is an interrupt function that includes a function call.

3-2. The problematic function is an interrupt or ordinary function that includes a parameter or automatic variable, and the parameter or automatic variable is internally assigned to the HL register[Note].

Note If the `-qr` option is specified, parameters or automatic variables might be assigned to general registers other than the HL register or to the internal memory. This can be directly checked in the assembler source or by using a tool.

Non-applicable example:

```
__interrupt void func1() {
             typedef int T1;
             func6();
}
```

The restriction does not apply because there is a function call in an interrupt function.

```
void func2() {
       struct S1 {
            int i;
      } s1;
}
```

The restriction does not apply because the automatic variable `s1` is created in the frame.

```
int gi;
  void func3() {
           union U1 {
                int i;
          };
           gi = TOE0L.3;
  }
```

The restriction does not apply if code that references the HL register exists in the function.

Workaround:

Declare `typedef`, `struct`, `union`, or `enum` outside the function even if no interrupt function (`_interrupt`) is specified.

Example 1:

```
typedef int T1;     /* Declare typedef outside the function */
__interrupt void func1() {
}


struct S1 {         /* Declare struct outside the function */
     int i;
 };
 __interrupt void func2() {
   }


 union U1 {         /* Declare union outside the function */
         int i;
 };
 __interrupt void func3() {
   }


 enum E1 {          /* Declare enum outside the function */
    E11
 };
__interrupt void func4() {
}
```

Example 2:

```
union U2 {          /* Declare union outside the function */
     int i;
 };
__interrupt void func5() {
             static union U2 u2;
}
```

Correction:

This issue has been corrected in V1.20.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 21 Invalid code is output when performing indirect reference if the `-ra` or `-rc` option is specified.

Description:

Invalid code is output if all the following conditions are satisfied:

(1) The `-ra` or `-rc` option is specified.

(2) An object pointer is pointed to by another pointer and the object is allocated to a `far` area (or the `far` qualifier is removed and the `-ml` option is specified).

(3) The indirectly referenced variable is assigned to the object pointer, and reused in the same expression.

Example 1:

```
[.c]
__far char **p;
__far char *cp;
void func(void)
{
        cp = ++(*p);
}
```

The address pointed to by `++(*p)` is invalid.

The value to be assigned to the variable `cp` is incorrect.

```
[ .asm ]
; line  5 :      cp = ++(*p);
 movw   de, !_p
 mov    a, [de+2]
 mov    _@SEGAX, a
 movw   ax, [de]
 incw   ax
 movw   [de], ax
                                ; xch  a, x  is not output.
 movw   !_cp, ax
 mov    a, _@SEGAX
 mov    !_cp+2,a
 ----------------------
```

Example 2:

```
[.c]
void func(void)
{
        __far char **tAu;
        *tAu[0]++ = 1;
}
------------------------
```

Because the address pointed to by `*tAu[0]++` is invalid, the value is assigned to an invalid address.

Workaround:

Do either of the following:

(1) Write the assignment expression and prefix/postfix increment/decrement expression separately.

(2) Disable the `-ra` and `-rc` options.

Workaround for *Example 1*:

```
[.c]
__far char **p;
__far char *cp;
void func(void)
{
        ++(*p);
        cp = (*p);
}
------------------------
```

Workaround for *Example 2*:

```
[.c]
void func(void)
{
        __far char **tAu;
        *tAu[0] = 1;
        tAu[0]++;
}
------------------------
```

Correction:

This issue has been corrected in V2.00.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 22 The ES setting code stored in an array or pointer might not be output.

Description:

The ES setting code stored in an array or pointer might not be output if any of the following conditions 1, 2 and 3 are satisfied.

- Condition 1:
  - (1) The left operand of a binary expression is an `int` or `unsigned int`.
    (This restriction does not apply if the operand is a variable, or if the expression uses pointers or arrays.)
  - (2) The right operand of the binary expression is an `unsigned int` type `far` array (whose address cannot be determined statically), or an `unsigned int` type `far` pointer.
  - (3) The used binary operator is `+`, `-`, `&`, `|`, or `^`.
    (For `+`, `&`, `|` and `^`, this condition also applies if the left and right operands are swapped.)
  - (4) No stack variable is included or no variable assigned to the HL register is included.

(This condition does not apply if the HL register is used as a stack frame pointer or used to store register variables.)

Example for *Condition 1*:

```
[.c]
unsigned char __far ar1[5];
unsigned int idx1, x1, x2;
void func(void)
{
        x1= (x2 << 1) + ar1[idx1] ;
}
------------------------


[ .asm ]
; line  5 :       x1 = (x2 << 1) + ar1[idx1];
movw    ax, !_idx1
addw    ax, #loww (_ar1)
movw    hl, ax
movw    ax, !_x2
addw    ax, ax
xch     a, x
                          ; mov  ES,#highw (_ar1) is not output
and     a, ES:[hl]
xch     a, x
addc    a, #00H           ; 0
movw    !_x1, ax
   ----------------------
```

• Condition 2:

(1) The value of an operand on one side of a relational expression can be determined statically, and the combination of the value and used relational operator is any of the following:

Operator	Type	Value[Note]

>, <=	`signed char, signed short`
	`signed int, signed long`
		–1

	`unsigned char`	0x7f
	`unsigned int, unsigned short`	0x7fff
	`unsigned long`	0x7fffffff

>=, <	`signed char, signed short`
	`signed int, signed long`
		0

	`unsigned char`	0x80
	`unsigned int, unsigned short`	0x8000
	`unsigned long`	0x80000000

Note Assumes an integer value is the right operand. If it is the left operand, the operator is reversed.

(2) The operand on the other side of the relational expression is an `int` type `far` array (whose address cannot be determined statically) or a `far` pointer.

(3) No stack variable is included or no variable assigned to the HL register is included. (This condition does not apply if the HL register is used as a stack frame pointer or used to store register variables.)

• Condition 3:

(1) The value of an operand on one side of an equality expression can be determined as 0 statically.

(2) The operand on the other side of the equality expression is a structure including bit field members, a `union` type `far` array (whose address cannot be determined statically), or a `far` pointer.

(3) No stack variable is included or no variable assigned to the HL register is included. (This condition does not apply if the HL register is used as a stack frame pointer or used to store register variables.)

Example for *Conditions 2* and *3*:

```
[.c]
struct st {
        int b0:1;
        int b1:2;
} __far *pbf;
int __far *p;
unsigned char uc;
unsigned int  x1;
void func(void)
{
    x1= (x1 + 1) - ((x1 + 1) - (*p < 0)) ;                   /* Condition 2 */
    x1= (x1 + 1) - ((x1 + 1) - (pbf->b0 == (uc = 0))) ;       /* Condition 3 */
}
```

Workaround:

Define a dummy automatic variable in the problematic function.

Example for *Condition 1*:

```
[.c]
unsigned char __far ar1[5];
unsigned int idx1, x1, x2;
void func(void)
{
        unsigned char dummy;        /* Define a dummy automatic variable */
        x1= (x2 << 1) + ar1[idx1];
 }
```

Correction:

This issue has been corrected in V2.00.
A tool used to check whether this restriction applies is available.
For details, contact a Renesas Electronics distributor.

No. 23 Invalid code is output if the last element of an initializer list for a `char`, `signed char`, or `unsigned char` array is a character string and one or more constants or character constants are placed before the character string.

Description:

If the last element of an initializer list for a `char`, `signed char`, or `unsigned char` array is a character string and one or more constants or character constants are placed before the character string, no error is output, but invalid code might be output.
Only string literals or constants can be used to initialize `char` or `unsigned char` arrays.

Example:

```
[.c]
const char a1[ ] = { 0x01, "abc" };
char *const TBL[3] = { a1 };
char *const *ptr1;
void func()
{
     ptr1 = TBL;
     **ptr1 = 0x12
}
 -------
[.asm]
@@CNST         CSEG   MIRRORP
_a1:  DB       01H    ; 1
      DB       'ab'
TBL:  DW       low (_a1)          ; _TBL is an odd address
      DB       (4)

@@DATA         DSEG   BASEP
_ptr1:         DS     (2)

; line     1 : const char a1[] = { 0x01, "abc" };
; line     2 : char *const TBL[3] = { a1};
; line     3 : char *const *ptr1;
; line     4 : void func()
; line     5 : {

@@CODEL        CSEG
_func:
; line     6 :        ptr1 = TBL;
      movw     ax,#loww (_TBL)    ; _TBL is an odd address
```

```
            movw      !_ptr1,ax
      ; line      7 :         **ptr1 = 0x12;
            movw      de,ax
            movw      ax, [de]              ; Reference an odd address
            movw      de,ax
            mov       [de+0], #012H
```

Workaround:

Correctly specify the initial value.

```
[.c]
const char a1[ ] = { 0x01, 'a', 'b', 'c', '\0' };
char *const TBL[3] = { a1 };
char *const *ptr1;
void func()
{
ptr1 = TBL;
**ptr1 = 0x12;
}
```

Correction:

This issue has been corrected in V2.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 24 Invalid code is output if referencing a pointer that has an offset obtained by subtracting one pointer from another.

Description:

Invalid code is output if all the following conditions are satisfied:

(1) A pointer to which an offset is added is referenced.

(2) The offset mentioned in (1) is the result of subtracting one pointer from another.

(3) A pointer mentioned in (2) has an offset.

Example:

```
[.c]
void main(void)
{
     char  *p1;
     char  *p2;
     char  *p3;

     *p1 = *(p2 + (p1 - (p3 + 2)));
}
```

Workaround:

Divide the expression.

```
[.c]
     void main(void)
{
     char  *p1;
     char  *p2;
```

```
          char  *p3;
          int tmp;                 /* Prepare a temporary variable */

          tmp = (p1-(p3 + 2));     /* Assign the value to the temporary variable */
          *p1 = *(p2 + tmp);
      }
```

Correction:

This issue has been corrected in V2.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 25 Invalid code is output if the `-qc` option has not been specified (sign extension is specified to be `int` type).

Description:

Invalid code is output if all the following conditions are satisfied:

(1) The `-qc` option is not specified (sign extension is specified to be the `int` type)

(2) One of the following combinations of operands (regardless of whether they are right or left operands) is multiplied:

- An `unsigned char` to which a constant from 0 to 255 is assigned and a constant from 0 to 255
- More than one `unsigned char` to which a constant from 0 to 255 is assigned
- An `unsigned char` to which a constant from 0 to 255 is assigned and a `char` or `signed char` to which a constant from 0 to 127 is assigned

(3) The multiplication result is 256 or larger (which cannot be expressed as an `unsigned char`).

(4) The operation result is handled as an `int`.

In the following example, the value of `Temp1` should be `0x1FE` but it is output as `0xFE`.

Example:

```
[.c]

unsigned int   Temp1;
unsigned char  Byte1;

Temp1 = (Byte1 = 255) * 2;
```

Workaround:

Cast the variable to an `int` or `unsigned int`.

```
[.c]

unsigned int   Temp1;
unsigned char  Byte1;

Temp1 = (unsigned int) (Byte1 = 255) * 2;
```

Correction:

This issue has been corrected in V2.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 26 Invalid code is output as a result of the BCD operation function `adbcdw` or `sbbcdw`.

Description:

Invalid code is output if one of the following conditions is satisfied while the BCD operation function `adbcdw` or `sbbcdw` is used:

(1) The return value of `adbcdw` or `sbbcdw` is assigned to an array or pointer.

(2) Another operation is executed before assigning the return value of `adbcdw` or `sbbcdw` to a temporary variable.

(3) A temporary variable to which the return value of `adbcdw` or `sbbcdw` has been assigned is used as is for other operations such as a conditional expression.

Example:

```
[.c]

void func()
{
     unsigned int   tmp1[3];
     unsigned int   tmp2, tmp3;
     unsigned int   a = 10, i = 0, *p;

     tmp1[i] = adbcdw(80, 50);                    /* (1) */
     tmp2 = adbcdw(80, 50) + (a + 1);             /* (2) */

     if ((tmp3 = adbcdw(80, 50) == *p ) …        /* (3) */
               ...
```

Workaround:

Prepare a function used only for calling `adbcdw` and `sbbcdw` and call the function. Use the same parameters and return values as those of `adbcdw` and `sbbcdw`.

Example:

```
[.c]

unsigned int adbcdw_new(unsigned int a, unsigned int b)
{
       return adbcdw(a, b);
}
void func()
{
   unsigned int   tmp1[3];
   unsigned int   tmp2, tmp3;
   unsigned int   a = 10, i = 0, *p;

   tmp1[i] = adbcdw_new(80, 50);                    /* (1) */
   tmp2 = adbcdw_new(80, 50) + (a + 1);             /* (2) */

   if ((tmp3 = adbcdw_new(80, 50) == *p ) …        /* (3) */
               ...
```

Correction:

This issue has been corrected in V2.10.

No. 27 An error occurs if the `-ng` option is specified and a branch instruction is used in a function that includes the `asm` statement.

Description:

If the `-ng` option is specified and a branch instruction is used after the `asm` statement in a function, an error might occur. This error might occur if both of the following conditions are satisfied:

(1) The `asm` statement is used in a function.

(2) A statement that causes processing to branch (such as `if`, `for`, or `while`) is used in the same function.

However, if the above conditions are satisfied, the error occurs during assembly and the object module file is not generated. If no error occurs, this restriction does not apply.

Example:

```
[.c]
unsigned int i;
void func()
{
     do {
         __asm("\t DB  (1000)");
         i++;
     } while ( i < 10) ;
}
```

Workaround:

Change to the `-g` option.

Correction:

This issue has been corrected in V2.12.

No. 28 The line number is not output for a statement that immediately follows a nested `if` statement and is outside that statement's conditional block.

Description:

If all of the conditions below are satisfied, the line number might not be output for a statement that immediately follows a nested `if` statement and is outside that statement's conditional block. However, the output code is correct. A break point cannot be specified for a statement for which the line number is not output.

(1) There are at least three levels of nested `if` statements.

(2) An `else` statement in a higher nested level has a larger line number than a nested `if` statement.

(3) At least one statement immediately follows an `if` statement in a higher nested level.

Example:

```
  [.c]
  int  f0, f1, f2, f3;
  int  g0, g1, g2;
  void func(void)
{
      if (f0){
          if (f1){     /* if statement in nested level 1 */
             g2 = 5;
          }
          else if (f2){/* if statement in nested level 2 */
              g2 = 4;
          }
          else if (f3){ /* if statement in nested level 3 (condition (1)) */
             g2 = 3;
          }
          else {
             g2 = 2;
          }
          g0 = 0x1234;  /* A statement immediately follows the if statement
                           in nested level 1 (condition (3)).*/
          g1 = 0x5678;
      }
      else {           /* This else statement has a larger line number than */
          g2 = 1;      /* the if statement in nested level 3 (condition (2)).*/
      }
}
```

Workaround:

Insert several blank lines before the statement that immediately follows the nested `if` statement and is outside that statement's conditional block.

For the above example, insert the lines before `g = 0x1234;`.

Correction:

This issue has been corrected in V2.12.

No. 29 Restriction due to an instruction (`mov`, `movw`) that uses `word[BC]` as an operand accessing the incorrect address

Description:

In either of the cases below, the output code is invalid.

For the `word[BC]` operand, if the `word` + `BC` address exceeds 10000H, an unintended address is accessed.

(1) Indirect reference is performed by subtracting a constant from a pointer.

However, this does not apply in the following cases:

(a) When the large model is used and the pointer is not pointing to a `near` area

(b) When the pointer is pointing to a `far` area

(c) When the pointer is pointing to one byte of data and the constant 1 is subtracted

(d) When the pointer is pointing to one byte of data, the constant 2 is subtracted, and optimization is not being performed with speed specified as having priority[Note].

(e) When the pointer is pointing to two bytes of data, the constant 1 is subtracted, and optimization is not being performed with speed specified as having priority[Note].

Note When `-qx1` is specified, or `l` is not specified for the `-q` option

(2) When referencing an array element, the variable used as the index contains a negative value, or indirect reference is performed by adding an offset (including a variable that contains a negative value) to the address indicated by an aggregate[Note].

Note An aggregate is a structure or array.

Condition (1) example

```
-------*.c-------------
unsigned char x[5], *ucp1;
unsigned short y[5], *usp1;
signed short si1;
void func1()
{
        /*********When the pointer accesses one byte of data*********/
x[0] = *(ucp1 - 1);   /* No problem */
x[1] = *(ucp1 - 2);   /* A problem occurs only if speed is prioritized for optimization. */
x[2] = *(ucp1 - 3);   /* A problem occurs. */
x[3] = ucp1[-4];      /* A problem occurs. The code has the same meaning as *(ucp1 -4). */
x[4] = *(ucp1 + si1 - 5); /* A problem occurs.*/
        /*********When the pointer accesses two bytes of data*********/
y[0] = *(usp1 - 1);   /* A problem occurs only if speed is prioritized for optimization. */
y[1] = *(usp1 - 2);    /* A problem occurs. */
y[2] = *(usp1 - 3);    /* A problem occurs. */
y[3] = usp1[-4];       /* A problem occurs. The code has the same meaning as *(usp1 -4). */
y[4] = *(usp1 + si1 - 5); /* A problem occurs. */
}
--------*.asm--------- (Excerpt from the assembly code for the above)
; line    25 :   x[2] = *(ucp1 - 3);  /* NG */
$DGL   0,9
       movw  bc,!_ucp1       ;[INF] 3, 1
       mov   a,65533[bc]     ;[INF] 3, 1 ← An invalid address might be accessed.
       mov   !_x+2,a         ;[INF] 3, 1
                          ...
---------------------
```

Condition (2) example

```
unsigned char x[4], *ucp1, uca1[10];
signed short sidx;
signed char cidx;
void func2()
{
 x[0] = uca1[cidx];       /* A problem only occurs if cidx is a negative non-zero value. */
 x[1] = *(&uca1[3] + cidx); /* A problem only occurs if cidx is a negative non-zero value. */
 x[2] = uca1[sidx];        /* A problem only occurs if sidx is a negative non-zero value. */
 x[3] = *(&uca1[4] + sidx); /* A problem only occurs if sidx is a negative non-zero value. */
}

--------*.asm--------- (Excerpt from the assembly code for the above)
; line    36 :      x[0] = uca1[cidx];       /* cidx must be a positive value. */
$DGL  0,20
      mov     a,!_cidx       ;[INF] 3, 1
```

```
        sarw    ax,8            ;[INF] 2, 1
        movw    bc,ax           ;[INF] 1, 1
        mov     a,_uca1[bc]     ;[INF] 3, 1 ← An invalid address might be accessed.
        mov     !_x,a           ;[INF] 3, 1
---------------------
```

Workaround:

- Workaround for condition (1)

  If indirect reference is performed when subtracting a constant from a pointer, assign the result of the subtraction to a different pointer before accessing an address.

Example 1:

Before applying the workaround	After applying the workaround

`unsigned char x, *ucp1;`
`void func1()`
`{`
   `x = *(ucp1 - 3);`
 	
`}`
	`unsigned char x, *ucp1,`
	`*ucp2;`
	`void func1()`
	`{`
	   `ucp2 = ucp1 - 3;`
	   `x = *ucp2;`
	`}`

Example 2:

Before applying the workaround	After applying the workaround

`unsigned char x, *ucp1;`
`void func2()`
`{`
   `x = ucp1[-4];`
 	
`}`
	`unsigned char x, *ucp1,`
	`*ucp2;`
	`void func2()`
	`{`
	 `ucp2 = ucp1 - 4;`
	 `x = *ucp2;`
	`}`

- Workaround for condition (2)

  If a variable that has a negative value is used in a signed index expression to reference an array element, or if indirect reference is performed by adding an offset (a variable that has a negative value) to an address that indicates an aggregate, assign the result of calculating the expression to a pointer before accessing an address.

Example 1:

Before applying the workaround	After applying the workaround

`unsigned char x, uca1[10];`
`signed char cidx;`
`void func1()`
`{`
   `x = uca1[cidx];`
 	
`}`
	`unsigned char x, uca1[10], *ucp1;`
	`signed char cidx;`
	`void func1()`
	`{`
	   `ucp1 = uca1 + cidx;`
	   `x = *ucp1;`
	`}`

Example 2:

Before applying the workaround	After applying the workaround

`unsigned char x, uca1[10];`
`signed char cidx;`
`void func2()`
`{`
  `x = *(&uca1[3] + cidx);`
 	
`}`
	`unsigned char x, uca1[10], *ucp1;`
	`signed char cidx;`
	`void func2()`
	`{`
	  `ucp1 = &uca1[3] + cidx;`
	  `x = *ucp1;`
	`}`

Correction:

This issue has been corrected in V2.13.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 30 The C0101 error occurs when the `-qj` option is specified

Description:

The C0101 error occurs if both of the following conditions are satisfied:

(1) The jump optimization option (`-qj`) is enabled.

(2) Any of the following code is included in processing that is not executed, such as an `if(0)` or `if(1)` else statement:

- Initialization for arrays, structures, or unions
- `switch` statements accompanying table jumps

Example:

```
[*.c]
struct t1 {
unsigned char uc1;
};
char c;
void func1()
{
  struct t1 st0 = {0x07};
    if(0) {
             struct t1 st0 = {0x08};    /* Processing that is not executed */
             c++;
    }
}
```

Workaround:

Do either of the following:

(1) Disable the `-qj` option.

(2) Delete the code that is not executed or embed the code between `#if 0` and `#endif` so as not to make it subject to compiling.

Correction:

This issue will be corrected in V2.30.

No. 31 Restriction on type-casting addresses in the `far` area to a `long` or `unsigned long`

Description:

If an address in the `far` area is type-cast to a `long` or `unsigned long` and the result is used as the initial value of the address, the highest byte of the address is fixed to 0FH.

Example:

```
[*.c]
__far int const fi1 = 5;
__far unsigned long const ula1[] = { &fi1 };
__far unsigned long const ula2[] = { (unsigned long)&fi1 };

[*.asm]
@@CNSTL  CSEG   PAGE64KP
_fi1:    DW     05H    ; 5
_ula1:   DG     _fi1
_ula2:   DW     loww (_fi1)
        DW     0FH    ; 15
```

Workaround:

Do not type-cast an address to a `long` or `unsigned long` if the initial value is used for the address.

Correction:

This issue will be corrected in V2.30.
A tool used to check whether this restriction applies is available.
For details, contact a Renesas Electronics distributor.

No. 32 Macro expansion using the `##` operator results in an error

Description:

The E0803 error might occur if the `##` operator is not followed by a function-like macro parameter, capital or small letter, or underscore (_).
In addition, using the `##` operator might cause errors other than E0803, such as E0711 or E0301.

Example 1:

```
[*.c]
#define m1(x) (x ## .c1 + 23)
#define m2(x) (x ## .c1 + 122)
struct t1 {
   unsigned char c1;
} st1;
unsigned char x1, x2;
void func1()
{
          x1 = m1(st1) + 100;    /* E0803 error (NG) */
          x2 = m2(st1) + 1;      /* No error   (OK) */
}
```

Example 2:

```
[*.c]
#define m3(x) (x ## 1)
unsigned char x3, uc1;
void func2()
{
          x3 = m3(uc);            /* E0711, E0301 error (NG) */
}
```

Workaround:

Do either of the following:

(1) Do not use the `##` operator.

```
#define m1(x)   (x ## .c1 + 23)
#define m2(x)   (x ## .c1 + 122)
    ↓
#define m1(x)   ((x).c1 + 23)
#define m2(x)   ((x).c1 + 122)
```

(2) Place a function-like macro parameter immediately after the `##` operator.

```
#define m3(x)   (x ## 1)
unsigned char x3, uc1;
void func2()
{
    x3 = m3(uc);
}
    ↓
#define m3(x, y)  (x ## y)
unsigned char x3, uc1;
void func2()
{
    x3 = m3(uc, 1);
}
```

Correction:

This issue will be corrected in V2.30.

No. 33 Symbol information for an interrupt function is not output to the assembler source

Description:

The E3405 error occurs during linking if all the following conditions are satisfied:

(1) `#pragma interrupt` is used to specify the generation of a vector table for an interrupt function.

(2) The interrupt function is not defined in the same source.

(3) The `-no` option, assembler source module file output option (`-a` or `-sa`), and debugging information output option (`-g`) are enabled.

Example:

```
[*.c]
#pragma interrupt INTP0 inter
/*     Definition of this interrupt function is not subject to compilation
__interrupt void inter()
{
      ...
}
*/
```

Workaround:

Define the interrupt function or output the object module file.

Correction:

This issue will be corrected in V2.30.

No. 34 Code that specifies the ES register settings might not be output

Description:

Code that specifies the ES register setting might not be output if a `far` variable is referenced directly after a floating point variable is incremented or decremented.

Example:

```
[*.c]
__far char c1;
float f1;
void func()
{
        ++f1;
        c1 = 5;
}
```

Workaround:

Immediately after incrementing or decrementing a floating point variable, insert a dummy code that references the variable by using a pointer.

```
[*.c]
__far char c1;
float f1;
void func()
{
        char *cp1;          /* Dummy variable */
        ++f1;
        *cp1;               /* Dummy dereference */
        c1 = 5;
}
```

Correction:

This issue will be corrected in V2.30.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.