[Notes]
C Compiler Package for RL78 Family
(CCRL#029-CCRL#032)

R20TS0793EJ0100
Rev.1.00
Jan. 16, 2022

Outline

When using the CC-RL C Compiler Package for the RL78 Family, note the following points.

1. Use of struct/union type arguments (CCRL#029)
2. Cast from pointer type to other type (CCRL#030)
3. Use of an anonymous struct/union (CCRL#031)
4. Use of an address read from memory after writing the address to the memory (CCRL#032)

Note: The number following the note is an identification number for the note.

1. Use of Struct/Union Type Arguments (CCRL#029)

1.1 Applicable Products

CC-RL V1.01.00 to V1.10.00

1.2 Details

An incorrect code may be generated when a struct or union type argument with a pointer-type member is used.

There are two conditions in which the problem may occur. Refer to the following for the conditions, examples, and the workarounds.

1.3 Conditions (1)

The problem may occur when all the following conditions are met.

(1) The option *-Onothing* is not specified.

(2) There is a struct or union type variable with a near-pointer-type member.

(3) The size of the struct or union type of (2) is 4 bytes or less.

(4) There is Function A that does (4-1) to (4-3) below.

(4-1) It assigns an address to the pointer-type member of (2).

(4-2) It passes the variable of the struct or union type, to which (4-1) has been assigned, to the argument and calls Function B.

(4-3) It references the destination of the address assigned in (4-1).

(5) Function B receives the struct or union type argument of (4-2) from a register.

(6) In Function B, the destination of the pointer-type member of the argument of (4-2) is referenced.

(7) Function B is expanded inline.

[Example] ccrl -cpu=S3 -Ospeed tp.c (1), (7)

```
/* tp.c */
#include <stdio.h>
typedef struct {    // (2), (3)
  int* _pointer;
  int _value;
}myStruct;
```

```
int flg = 0;
void func( myStruct arg ) {      // (5)
  if (*(arg._pointer) != 10) {  // (6)
    flg = 1;
  }
}
void main(void) {
  int val = 10;
  volatile myStruct st;
  st._pointer = &val;      // (4-1)
  func(st);                // (4-2)
  val = 20;                // (4-3)
  if (flg == 1) {
    printf("ng¥n");
  } else {
    printf("ok¥n");
  }
}
```

In this example, the variable *flg* should not become 1, and "*ok*" should be output by *printf*. However, the code that assigns 10 to the variable *val* has been deleted due to failure and "*ng*" is output as the result.

1.4 Workaround (1)

Do either of the following.

(a) Specify the option *-Onothing*.

(b) Change the size of the struct or union type to avoid condition (3) above.

(c) Adjust the struct or union type argument so that it is not passed by a register.

(d) Avoid the inline expansion.

(e) Change the pointer of condition (2) to a *far* pointer.

1.5 Conditions (2)

The problem may occur when all the following conditions are met.

(1) The option *-Onothing* is not specified.

(2) The option *-Ointermodule* is specified.

(3) There is a struct or union type with a near-pointer-type member.

(4) There is a function with the struct or union type argument of (3).

(5) In the function of (4), the pointer-type member of the struct or union type argument is referenced for either or both of the following.

(5-1) It has been read and written.

(5-2) It has been written multiple times.

[Example] ccrl -cpu=S3 -Ospeed -Ointermodule tp.c (1), (2)

```
/* tp.c */
typedef struct s_tag {  // (3)
    int *ptr ;
    int dmy1 ;
} STRCT ;
STRCT gv;
void func( STRCT arg ) { // (4)
    int i ;
    for( i = 0 ; i < 1 ; i++ ){
        (*(arg.ptr)) += 1 ;  // (5)
```

```
        (*(arg.ptr)) += 2 ;  // (5)
    }
    gv = arg;
}
```

In this example, 3 should be added to the destination of *arg.ptr*. However, the code that adds 2 to the destination is generated due to failure.

1.6 Workaround (2)

Do either of the following.

(a) Specify the option *-Onothing*.

(b) Unspecify the option *-Ointermodule*.

(c) Change the pointer of condition (3) to a *far* pointer.

1.7 Schedule for Fixing the Problem

The problem will be fixed in CC-RL V1.11.00, which will be released in January 2022.

2. Cast from Pointer Type to Other Type (CCRL#030)

2.1 Applicable Products

CC-RL V1.01.00 to V1.10.00

2.2 Details

An incorrect code may be generated when a casted pointer type is used.

2.3 Conditions

The problem may occur when all the following conditions are met.

(1) The option *-Onothing* is not specified.

(2) The option *-Ointermodule* is specified.

(3) A pointer-type value is casted to a non-pointer-type and then assigned to a variable.

(4) The address of the variable, to which (3) has been assigned, has been assigned to a pointer-to-pointer-type variable.*

(5) The destination of the pointer-to-pointer-type variable of (4) is referenced.

(6) The destination of the pointer-type variable assigned in (3) is referenced in the same function of (5).

*Instead of "a pointer-to-pointer-type", the condition also applies when there are multiple unary operators.

[Example] ccrl -cpu=S3 -Ospeed -Ointermodule tp.c (1), (2)

```
/* tp.c */
#include <stdio.h>
void test(int key){
   int gv = 0;
   volatile int variable = (int)&gv;   // (3)
   int** pointer = (int**)&variable;   // (4)
   **pointer = 1; // (5)

   if (gv == 1){   // (6)
      printf("ok");
   }
   else{
      printf("ng");
   }
```

```
}
```

In this example, "*ok*" should be output by *printf* as the result of assigning 1 to the variable *gv*. However, the *if-else* statement that compares the variable *gv* and 0 (assuming that *gv* is 0) has been deleted due to failure. As the result, "*ng*" is output.

2.4 Workaround

Do either of the following.

(a) Specify the option *-Onothing*.

(b) Unspecify the option *-Ointermodule*.

(c) Avoid casting a pointer-type value to a non-pointer-type, assigning it to a non-pointer-type variable, and referencing it via the variable of the pointer-to-pointer-type.

2.5 Schedule for Fixing the Problem

The problem will be fixed in CC-RL V1.11.00 , which will be released in January 2022.

3. Use of an Anonymous Struct/Union (CCRL#031)

3.1 Applicable Products

CC-RL V1.01.00 to V1.10.00

3.2 Details

A struct and union with an anonymous union type member may not be initialized correctly or cause internal errors.

3.3 Conditions

The problem may occur when all the following conditions are met.

(1) There is a variable or compound literal with either of the following types.

(1-1) Struct

(1-2) Union

(2) The type of (1) has either of the following members.

(2-1) Anonymous struct (Note 1)

(2-2) Anonymous union (Note 2)

(Note 1) A struct member with a member name omitted in the declaration

(Note 2) A union member with a member name omitted in the declaration

(3) The variable of (1) has been initialized with the declaration.

(4) Either of the following conditions is met.

(4-1) The unions of (1) and (2) have a member that is larger than the first member.

(4-2) A member smaller than the unions of (1) and (2) has been initialized in (3).

(4-3) The initialization of (3) has been done to only some members of the structs, unions, and classes of (1) and (2).

[Example] ccrl -cpu=S3 tp.c

```
/* tp.c */
long func() {
   struct { // (1-1)
      union { // (2-2)
         long a;
```

```
        };
        union {
           long b;
        } c;
    } v = {10}; // (3), (4-3)
    return v.c.b;
}
```

In this example, *v.a* (anonymous union member) should be initialized to 10, *v.c.b* to 0, and *func* should return 0 as the result. However, *v.c.b* is incorrectly initialized to 10, and *func* returns 10 as the result.

3.4 Workaround

Do either of the following.

(a) Separately perform the declaration and the initialization of the variables that meet condition (1).

(b) If condition (4-3) is met, initialize all the members of the variables that meet condition (1) at once.

(c) Avoid using anonymous structs and unions. Name and use them as structs and unions.

3.5 Schedule for Fixing the Problem

The problem will be fixed in CC-RL V1.11.00 , which will be released in January 2022.

4. Use of an Address Read from Memory After Writing the Address to the Memory (CCRL#032)

4.1 Applicable Products

CC-RL V1.01.00 to V1.10.00

4.2 Details

If a pointer has been written to memory and then read, an invalid code may be generated under certain circumstances.

4.3 Conditions

The problem may occur when all the following conditions are met.

(1) The option *-Onothing* is not selected.

(2) Regarding the memory, either of the following is met.

(2-1) An address has been written to the memory as a _near-pointer-type value. The value is then read as a non-pointer-type value, converted to a _near-pointer-type, and its destination is referenced.

(2-2) An address has been written to the memory as a non-pointer-type value. The value is then read as a _near-pointer-type value, and its destination is referenced.

(3) The destination of the address of (2) is referenced without using the read and write operations in (2).

(4) The references of (2) and (3) are in the same functions (or the references are put in a single function by inline expansion), and either of the references performs writing.

[Example] ccrl -cpu=S3 tp.c // (1)

```
/* tp.c */
#include <stdio.h>
int flg = 0;
void main(void){
    volatile union{
        int* _pointer;
        int _value;
    }myUnion;
    int val = 10;
```

```
    myUnion._pointer = &val;    // (2-1)
    if (*((int*)myUnion._value) != 10){ // (2-1)(4)
        flg = 1;
    }
    val = 20;            //(3)(4)
    if (flg == 1){
        printf("ng¥n");
    } else {
        printf("ok¥n");
    }
}
```

In this example, the variable *flg* should not become 1, and "*ok*" should be output. However, the code that assigns 10 to the variable *val* has been deleted due to failure, and "*ng*" is output as the result.

ccrl tp2.c -Ospeed -Ointermodule // (1)

```
/* tp2.c */
#include<stdio.h>
typedef struct{
    int *_pointer;
} MyStruct2;
void test(void){
    int autoVar = 0;
    volatile struct{
        int _value;
    } myStruct;
    volatile MyStruct2* castedStruct = (volatile MyStruct2*)(&myStruct);
    myStruct._value = (int)&autoVar;    // (2-2)
    *(castedStruct->_pointer) = 1; // (2-2)(4)
    if (autoVar == 1){ //(3)(4)
        printf("ok");
    }
    else{
        printf("ng");
    }
}
```

In this program, *autoVar* should become 1 and "*ok*" should be output. However, *autoVar* is judged to be 0 due to failure, the *if-else* statement is deleted, and "*ng*" is output as the result.

4.4 Workaround

Do either of the following.

(a) Specify the option *-Onothing*.

(b) When saving a pointer-type value, write it as a pointer type and read it as a pointer type.

(c) Change the pointer of condition (2) to a *far* pointer.

4.5 Schedule for Fixing the Problem

The problem will be fixed in CC-RL V1.11.00 , which will be released in January 2022.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jan.16.22	-	First edition issued

Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

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.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

TS Colophon 4.3