[Notes]
C/C++ Compiler Package for RX Family

R20TS0246EJ0100
Rev.1.00
Dec. 16, 2017

Outline

When using the C/C++ compiler package for the RX family V.1 to V.2, note the following points.

1. When a string is converted to a numeric value by using a standard library function (RXC-0046)
2. When a string operation library function is used or the suntil, smovu, or scmpu instruction is used in the assembler source (RXC-0047)

   Note: The number which follows the description of a precautionary note is an identifying number for the precaution.

1. When a String is Converted to a Numeric Value by Using a Standard Library Function (RXC-0046)

1.1 Applicable Products

All versions of C/C++ compiler package for the RX family

(V.1.00 Release 00 to V.2.07 Release 00)

1.2 Details

If the string of the first argument is converted to a numeric value by using a standard library function such as strtol(), a pointer that does not comply with the standard might be set in the second argument.

1.3 Conditions

This problem may arise if all of the following conditions are met.

(1) One of the following standard library functions is used.

- strtol()
- strtoll()
- strtoul()
- strtoull()
- strtod()

(2) A string that meets both Conditions (a) and (b), as follows, is specified in the first argument for (1).

(a) At least one white-space character(Note 1) exists at the beginning of the string.

(b) The white-space characters(Note 1) in (a) are followed by a character that does not represent an integer(Note 2).

Note 1: White-space characters include a space (' '), horizontal tab ('¥t'), line feed ('¥n'), vertical tab ('¥v'), form feed ('¥f'), and carriage return ('¥r').

Note 2: If the value of the third argument is from 10 to 36, some or all the alphabetic characters are handled as characters that represent an integer.

1.4 Example

Red text indicates the parts corresponding to the above conditions.

- Example source

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
long test;
const char* str = "   Hello World!"; /* (Conditions (2-a) (2-b)) */
void main(void) {
   char* endptr = NULL;
   test=strtol(str, &endptr, 0); /* (Condition (1)) */
   printf(" str=¥"%s¥"¥n endptr=¥"%s¥"¥n", str, endptr);
}
```

- Correct output result

According to the specifications, a pointer is set so that endptr is equal to str, and information is correctly output as follows.

```
 str="   Hello World!"
 endptr="   Hello World!"
```

- Actual output result

A pointer skipping the spaces is set, and information is incorrectly output as follows.

```
str="   Hello World!"
endptr="Hello World!"
```

Supplement: Standard of strtol()

The numeric part at the beginning of the string is converted to long type in the radix specified for base. If the string contains a character that cannot be converted, the pointer to that character is stored in endptr. If the space is followed by a numeric value, the data is skipped up to the end of the numeric value. However, if the space is followed by a non-numeric value, the data is stored in endptr without skipping the space.

For details about strtol(), refer to the URL below.

https://www.renesas.com/en-us/doc/products/tool/doc/009/r20ut3248ej0104_ccrx.pdf
CC-RX Compiler User's Manual
7.4 Library Function

How to Use the Function

```
long strtol(
    const char *nptr, /* String */
    char **endptr,
     /* Pointer to the storage area containing a pointer to the first character that does not represent an integer */
    int base /* Radix*/
);
```

Example source used

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
long test;
const char* str = "   123 Hello 456 World!";
void main(void) {
    char* endptr = NULL;
    test=strtol(str, &endptr, 0);
    printf(" str=¥"%s¥"¥n endptr=¥"%s¥"¥n test=%ld¥n",str,endptr,test);
}
```

Output result

```
 str="   123 Hello 456 World!"
 endptr=" Hello 456 World!" /* The address following the numeric value is stored in the pointer. */
 test=123/*The value converted as the return value is returned in long type. */
```

1.5 Workaround

To avoid this problem, use the following method.

Make sure that a white-space character(Note) is not contained at the beginning of the string that is passed to the first argument in Condition (1).

Note: White-space characters include a space (' '), horizontal tab ('¥t'), line feed ('¥n'), vertical tab ('¥v'), form feed ('¥f'), and carriage return ('¥r').

■ Example:

```
const char* str = "□□□ Hello World!"; /* Three spaces are contained. */
void main(void) {
    char* endptr = NULL;
    (void)strtol(&str[3], &endptr, 0);
    /* By prohibiting spaces, Condition (2) is avoided. */
        ...
```

1.6 Schedule for Fixing the Problem

The time to fix the problem is currently under consideration. Please use the workaround to resolve the problem.

2. When a String Operation Library Function is Used or the suntil, smovu, or scmpu Instruction is Used in the Assembler Source (RXC-0047)

2.1 Applicable Products

All versions of C/C++ compiler package for the RX family

(V.1.00 Release 00 to V.2.07 Release 00)

2.2 Details

When a string operation library function is used, or the suntil, smovu, or scmpu instruction is used in the assembler source, an access to storage areas that are not for strings may occur, resulting in incorrect operation.

The location of access to storage areas that are not for strings and the details of incorrect operation are as follows.

(1) When an access is made to a reserved area

An interrupt occurs due to an invalid address access error.

(2) When an access is made to an area protected by the MPU (Memory Protection Unit)

An exception processing occurs due to a data memory protection error.

2.3 Conditions

This problem may arise if the following conditions are all met.

(1) One of the string operation library functions shown in (1-1) is used. Or, an assembler instruction is used in the assembler source.

(1-1) String operation library functions

memchr()

strlen()

strcmp()

strncmp()

strcpy()

strncpy()

strcat()

strncat()

(1-2) Assembler instruction

suntil

smovu

scmpu

(2) Either of the following conditions is met.

(2-1) One of the string operation library functions in (1-1) is used.

The address(Note) of the string specified by the argument is not 4-byte alignment.

Note: The arguments to specify the address of a string are as follows.

String operation library functions	Arguments to specify the address of a string
memchr(const void *s, long c, size_t n);	s
strlen(const char *s)	s
strcmp(const char *s1, const char *s2)	s1, s2
strncmp(const char *s1, const char *s2, size_t n);	s1, s2
strcpy(char *s1, const char *s2)	s2
strncpy(char *s1, const char *s2, size_t n);	s2
strcat(char *s1, const char *s2)	s1, s2
memchr(const void *s, long c, size_t n);	s

(2-2) One of the assembler instructions in (1-2) is used.

The address(Note) of the string specified by the register is not 4-byte alignment.

Note: The register information to specify the address of a string is as follows.

Assembler instruction	Register information to specify the address of a string
suntil	Comparison target addresses indicated by the R1 register
smovu	Transfer destination addresses indicated by R1 are targeted.
scmpu	Comparison source addresses indicated by R1, Comparison target addresses indicated by R2

(3) When a string operation library function in (1) is used or the assembler instruction is used in the assembler source, the termination character ('¥0') of the string specified by the argument is located within 3 bytes immediately before a reserved area or area protected by the MPU.

2.4 Example

The following is an example of the problem.

➢ When calculating the number of characters ("ab") of string s using the string operation library function "strlen()"

(1) The allocation address of string s is address 0x0d which is not 4-byte alignment.

(2) The termination character ('¥0') is located within 3 bytes immediately before the reserved area.

Conditions (1) and (2) are satisfied, and an access is made to address 0x10 which is in the access prohibited area such as the reserved area or the area protected by the MPU.

Red text indicates the parts corresponding to the above conditions.

```
==========================================================
   size = strlen(s);       // Condition (1)

   Address 0x0d     Accessible area     'a'          The address of s in Condition (2) is 0x0d, which is not
                                                      4-byte alignment.

   Address 0x0e     Accessible area     'b'
   Address 0x0f     Accessible area     '¥0' (End condition) Condition (3)
   ----- 4-byte boundary ---
   Address 0x10     Access prohibited area (reserved area)
==========================================================
```

2.5 Workaround

To avoid this problem, do any of the following:

(1) Change the allocation of the string so that the target data of the string operation in Condition (2) is not located immediately before the reserved area or the area protected by the MPU.

(2) Use the -avoid_cross_boundary_prefetch option supported by CC-RX V2.07.00 or later.

2.6 Schedule for Fixing the Problem

The time to fix the problem is currently under consideration. Please use the workaround to resolve the problem.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec. 16, 2017	-	First edition issued

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061 Japan
Renesas Electronics Corporation

■Inquiry
https://www.renesas.com/contact/