Microcomputer Technical Information

CP(K), O

CC78K4 78K4 Series C Compiler Usage Restrictions	Document No.	ZBG-CD-04-0074	1/2
	Date issued	October 4, 2004	
	Issued by	Development Tool Group Multipurpose Microcomputer Systems Division 3rd Systems Operations Unit NEC Electronics Corporation	
Related documents CC78K4 Ver.2.40 or Later Operation: U16707EJ1 (1st) CC78K4 Ver.2.30 or Later Language: U15556EJ1 (1st) 78K4 Series C Compiler CC78K4 Ver.2.40 Operating Precautions: SUD-DT-04-0118	Notification classification	√	Usage restriction
			Upgrade
			Document modification
			Other notification

1. Affected product

   CC78K4 Ver. 2.40

2. New restrictions

   The following restrictions have been added in CC78K4 V2.40. See the attachment for details.

   - No. 23 An illegal code is output as a result of an operation that includes signed char type operator and constant values.
   - No. 24 An illegal code is output as a result of a compound assignment expression containing %= or /=.
   - No. 25 W503 is output when the array name of an automatic variable is referenced.

3. Workarounds

   The following workarounds are available for these restrictions. See the attachment for details.

   - No. 23 Cast the relevant constant as a signed int type.
   - No. 24 Use the simple assignment operator "=" instead of the compound assignment operators "%=" and "/=".
   - No. 25 When W503 is output, check the relevant location. Ignore the message if initialization has been performed in the statement.

4. Modification schedule

   No. 23 and No. 24 will be corrected in CC78K4 Ver. 2.50 (planned for release in February 2005).

   No. 25 is not planned for correction, so regard this item as a usage restriction.

   * For the detailed release schedule of modified products, contact an NEC Electronics sales representative.

5. List of restrictions

   A list of usage restrictions in the CC78K4, including the revision history and detailed information, is described on the attachment.

6. Revision history

CC78K4 78K4 Series C Compiler Usage Restrictions Revision History

Document Number	Date Issued	Description
ZBG-CD-04-0054	August 18, 2004	Newly created.
ZBG-CD-04-0074	October 4, 2004	Addition of restrictions No. 23 to No. 25

List of Usage Restrictions in CC78K4

1. Product History

No.	Bugs and Changes/Additions to Specifications	Version	
		V2.30	V2.40
1	If a character string includes a NULL character, the character string following the NULL character is invalid.	×	√
2	An illegal code is output when an array with an offset value is used in the code.	×	√
3	An illegal code may be output if the address of a function located in the flash memory memory area is referenced.	×	√
4	The constant expression of #if may not be processed correctly.	×	√
5	An illegal code may be output if the address of a multidimensional array is referenced and the same array is referenced after that.	×	√
6	An illegal code may be output if a variable used by a compound assignment operation of a bit-wise AND/XOR/OR is referenced by the following **switch** statement	×	√
7	An illegal code is output if a negative floating-point constant is cast to an unsigned integer type.	×	√
8	An illegal code is output if a logical OR or logical AND operation is performed between floating-point constants.	×	√
9	An illegal code is output if the function parameter is an array that is declared as a register	×	√
10	If variables with the same name are declared in multiple files while using the #pragma section, the variables may not be allocated to the correct section.	×	√
11	An illegal code is output if a logical OR or logical AND operation is performed between a floating-point constant and integer-type constant.	×	√
12	An illegal code may be output if a function that has a 5-byte or larger structure or union as an argument is called when using a large model	×	√
13	An illegal code may be output if a variable is used as the index of a structure array that has a bit field as a member and an assignment operation is performed on the bit field located at the higher byte of an array element.	×	√
14	When a conversion specifier of a decimal integer is used in the **scanf** or **sscanf** function and the value after converting a character string is 65536 or larger or −65536 or lower, an illegal result may be returned.	×	√
15	An illegal conversion result of a floating-point number may be output when the function **printf**, **sprintf**, **vprintf**, or **vsprintf** is executed in a medium model.	×	√
16	The initialization of an external variable declared extern within a block does not become an error. In addition, the debugging information in the assembler source is incorrect.	×	×
17	Binding a variable with the same name to a variable declared extern in the block is sometimes illegal.	×	×
18	If a type defined by typedef (typedef name) is used in a function prototype declaration or a declaration using a const or volatile type modifier, the typedef expansion is illegal, and an error results.	×	×

×: Applicable, √: Not applicable, –: Not relevant

No.	Bugs and Changes/Additions to Specifications	Version	
		V2.30	V2.40
19	Sometimes a multidimensional array with an undefined size does not operate properly.	×	×
20	In a function returning the address of a function with arguments, those arguments cannot be referenced. There is no error when referenced, but illegal code is output.	×	×
21	The signed type bit field is handled as an unsigned bit field.	×	×
22	If the total size of the auto variable in one function exceeds 65,535 bytes when the large model is used, the output code and debug information become illegal.	×	×
23	An illegal code is output as a result of an operation that includes signed char type operator and constant values.	×	×
24	An illegal code is output as a result of a compound assignment expression containing %= or /=.	×	×
25	W503 is output when the array name of an automatic variable is referenced.	×	×

×: Applicable, √: Not applicable, –: Not relevant

2. Details of Usage Restrictions

No. 1 If a character string includes a NULL character, the character string following the NULL character is invalid.

[Description]

If a character string includes a NULL character, the character string following the NULL character is invalid.

Example:

```
const char str[] = "test\0TEST";
```

No area is secured for the character string following the NULL character.

[Workaround]

Describe `const char str[] = {'t','e','s','t','\0','T','E','S','T'};`.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 2 An illegal code is output when an array with an offset value is used in the code.

[Description]

An illegal code is output if any of the following conditions (1) to (3) is satisfied when using a small model or medium model.

(1) An add-sub assignment operation is performed between an offset sreg/__sreg variable and a character string

(2) A conditional expression between **saddr** area data and an array, or an SFR symbol and an array is performed and the offset value of the array is 256 or larger

(3) When the argument _@NRARGx of a **norec** function is passed to an array whose offset value is 256 or larger

Example:

```
#pragma sfr
```

```
__leaf void nrfunc(int, int, int, int *);
__sreg struct {
    int a;
    int b;
} st;
int a[200], i;

void func()
{
    st.b -= (int)"abc";            /* (1) */
    if (CR00 == (int)&a[199]) i++; /* (2) */
    nrfunc(1, 2, 3, &a[199]);      /* (3) */
}
```

[Workaround]

Implement the following workarounds for conditions (1) to (3), respectively.

(1) Divide the description into an add-sub operation and an assignment operation.

```
st.b = st.b - (int)"abc";
```

(2) Insert a temporary variable, and execute the operation by assigning the operation result to the temporary variable.

```
int tmp1;
tmp1 = (int)&a[199];
if (CR00 == tmp1) i++;
```

(3) Insert a temporary variable, and execute the operation by assigning the operation result to the temporary variable.

```
int *tmp2;
tmp2 = &a[199];
nrfunc(1, 2, 3, tmp2);
```

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 3 An illegal code may be output if the address of a function located in the flash memory area is referenced.

[Description]

An illegal code may be output if the address of a function located in the flash memory area is referenced.

Example:

```
#pragma ext_table 0x2000
#pragma ext_func func1 1
#pragma ext_func func2 2

int func1(int, int);
int func2(int, int);
int (*func_b1)(int, int) = &func1;
__sreg int (*func_b2)(int, int) = &func2;

void func()
```

```
{
    func_b1 = func1;     /* normal code */
    func_b2 = func2;     /* illegal code */
}
```

[Workaround]

There is no workaround.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 4 The constant expression of #if may not be processed correctly.

[Description]

The constant expression of #if may not be processed correctly.

Example:

```
#define a
#if a
int i;
#endif
void func()
{
    i++;
}
```

[Workaround]

There is no workaround.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 5 An illegal code may be output if the address of a multidimensional array is referenced and the same array is referenced after that.

[Description]

An illegal code may be output if the address of a multidimensional array is referenced and the same array is referenced after that.

Example:

```
unsigned char a[10][1];
unsigned char *p,c,uc1,uc2;
void func()
{
    p = a[uc1];
    c = a[uc1][uc2];   /* illegal code */
}
```

[Workaround]

There is no workaround.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 6 An illegal code may be output if a variable used by a compound assignment operation of a bit-wise AND/XOR/OR is referenced by the following **switch** statement.

[Description]

An illegal code may be output if a variable used by a compound assignment operation of a bit-wise AND/XOR/OR is referenced by the following **switch** statement.

Example:

```
unsigned int ui, i;
void func(unsigned int p1, unsigned int p2)
{
    ui |= p2;
    switch (p2) {/* illegal code */
    case 0:
    case 1:
    case 2:
        i++;
    }
}
```

[Workaround]

Divide the function into a bit-wise AND/XOR/OR operation and a simple assignment operation.

```
ui = ui | p2;
```

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 7 An illegal code is output if a negative floating-point constant is cast to an unsigned integer type.

[Description]

An illegal code is output if a negative floating-point constant is cast to an unsigned integer type.

Example:

```
unsigned long ans;
float f1 = -3.8f;
void func()
{
    int a;
    ans = f1;
    a = ((unsigned long)(-3.8f) != ans);
}
```

[Workaround]

Cast a constant to a signed integer type before casting the constant to the unsigned integer type.

```
a = ((unsigned long)(long)(-3.8f) != ans);
```

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 8 An illegal code is output if a logical OR or logical AND operation is performed between floating-point constants.

[Description]

An illegal code is output if a logical OR or logical AND operation is performed between floating-point constants.

Example:

```
void func()
{
    int r1, r2;
    float f1 = 17, f2 = 16;
    r1 = f1 || f2;
    r2 = f1 && f2;
}
```

[Workaround]

There is no workaround.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 9 An illegal code is output if the function parameter is an array that is declared as a register.

[Description]

An illegal code is output if the function parameter is an array that is declared as a register.

Example:

```
void func(register char a[])
{
    register char *p;
    p = (char *)a;
}
```

[Workaround]

Describe a pointer type, instead of describing the array.

```
void func(register char *a)
{
    register char *p;
    p = a;
}
```

No. 10 If variables with the same name are declared in multiple files while using the #pragma section, the variable may not be allocated to the correct section.

[Description]

If variables with the same name are declared in multiple files while using the #pragma section, the variable may not be allocated to the correct section.

Example:

```
--- test.c ---
#include "a.h"
#include "b.h"
#include "c.h"

--- a.h ---
#pragma section @@DATA DAT1
int i;
```

```
#pragma section @@DATA DAT2

--- b.h ---
#pragma section @@DATA DAT3
int j;
#pragma section @@DATA DAT4

--- c.h ---
#pragma section @@DATA DAT5
extern int i;          /* same when int i;  */
#pragma section @@DATA DAT6
```

[Workaround]

Do not use variables with the same name in multiple files when using the #pragma section.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 11 An illegal code is output if a logical OR or logical AND operation is performed between a floating-point constant and integer-type constant.

[Description]

An illegal code is output if a logical OR or logical AND operation is performed between a floating-point constant and integer-type constant.

Example:

```
void func()
{
     int rval;
     int cZero = 0;
     rval = 0.0F || cZero;    /* illegal code in Windows version */
     rval = cZero || 0.0F;    /* illegal code in UNIX version */
}
```

[Workaround]

Do not describe a floating-point constant for the logical OR or logical AND operation.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 12 An illegal code may be output if a function that has a 5-byte or larger structure or union as an argument is called when using a large model.

[Description]

An illegal code may be output if a function that has a 5-byte or larger structure or union as an argument is called when using a large model.

Example:

```
struct {
    int a;
    int b;
    int c;
} st;
void (*fp)();
```

```
void func()
{
    (*fp)(st);
}
```

[Workaround]

There is no workaround.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 13 An illegal code may be output if a variable is used as the index of a structure array that has a bit field as a member and an assignment operation is performed on the bit field located at the higher byte of an array element.

[Description]

An illegal code may be output if a variable is used as the index of a structure array that has a bit field as a member and an assignment operation is performed on the bit field located at the higher byte of an array element.

Example:

```
struct st {
   unsigned int bit0:1;
   unsigned int bit1:1;
   unsigned int bit2:1;
   unsigned int bit3:1;
   unsigned int bit4:1;
   unsigned int bit5:1;
   unsigned int bit6:1;
   unsigned int bit7:1;
   unsigned int bit8:1;
   unsigned int bit9:1;
   unsigned int bitA:1;
   unsigned int bitB:1;
   unsigned int bitC:1;
   unsigned int bitD:1;
   unsigned int bitE:1;
   unsigned int bitF:1;
} str[5];
unsigned char num;

void func()
{
    str[num].bit8 = 1;  /* Illegal code */
}
```

[Workaround]

Insert a temporary variable, and execute the operation by assigning the operation result to the temporary variable.

```
struct st *tmp;
tmp = &str[num];
```

```
tmp->bit8 = 1;
```

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 14 When a conversion specifier of a decimal integer is used in the **scanf** or **sscanf** function and the value after converting a character string is 65536 or larger or –65536 or lower, an illegal result may be returned.

[Description]

When a conversion specifier of a decimal integer is used in the **scanf** or **sscanf** function and the value after converting a character string is 65536 or larger or –65536 or lower, an illegal result may be returned.

Internally, a character string is replaced by numbers sequentially from the lower digit.

(1) One character is acquired and converted into a numeric value.
(2) If all the input characters are converted, the program jumps to (7).
(3) The radix is multiplied (carry operation).
(4) The next character is acquired and converted into a numeric value.
(5) The converted value is added to the multiplied result (processing relevant to this bug)
(6) The program jumps to (2).
(7) End

An illegal value is returned if the result of addition of the lower 2 bytes in step (5) exceeds 65535.

Example:

```
long a;
char buf[16];

void func()
{
    strcpy(buf, "65536");
    sscanf(buf, "%ld", &a);
}
```

[Workaround]

Implement any of the following workarounds.

(1) Describe both the character string and conversion specifer using the octal or hexadecimal expression.

Example:

```
strcpy(buf, "0x10000");
sscanf(buf, "%lx", &a);
```

(2) Use the **atol** function to acquire the value from the character string.

Example:

```
#include<stdlib.h>
a = atol(buf);
```

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 15 An illegal conversion result of a floating-point number may be output when the function **printf**, **sprintf**, **vprintf**, or **vsprintf** is executed in a medium model.

[Description]

An illegal conversion result of a floating-point number may be output when the function **printf**, **sprintf**, **vprintf**, or **vsprintf** is executed in a medium model.

Example:

```
#include <stdio.h>
void func()
{
    char res[20];
    sprintf(res, "%f", 1.2e-38);
}
```

[Workaround]

There is no workaround.

[Correction]

This restriction has been corrected in Ver. 2.40.

No. 16 The initialization of an external variable declared extern within a block does not become an error. In addition, the debugging information in the assembler source is incorrect.

[Description]

Since it is not compliant with the ANSI C language specifications, the initialization of an external variable declared extern within a block should produce an error, but the description does not become an error. The object defined as an external variable with initial value is interpreted and the code is output by the compiler.

The debugging information in the object output by the compiler is correct, but the debugging information in the assembler source is incorrect.

Example:

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

[Workaround]

There is no workaround.

[Correction]

Regard this item as a usage restriction.

No. 17 Binding a variable with the same name to a variable declared extern in the block is sometimes illegal.

[Description]

Binding a variable with the same name to a variable declared extern in the block is illegal in either of the following cases.

(1) When a variable declared with extern in a block and a variable declared with static after outside the block have the same name

Since no error occurs and there is no binding, illegal code is output when this variable is

referenced.

Example:

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

(2) When a variable declared with extern in a block and a variable not declared with static outside the block after a variable declared with extern have the same name

There is no binding, and illegal code is output.

Example:

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

(3) When a variable declared with extern in a block and a variable not declared with extern outside the block before a variable declared with extern have the same name, and an automatic variable declared in a block containing the block with the variable declared with extern has the same name

The variable outside the block and the variable declared with extern in the block are not bound, and illegal code is output.

Example:

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

(4) A variable declared with extern in a block and a variable declared with extern in another block have the same name

There is no binding, and illegal 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]

Regard this item as a usage restriction.

No. 18 If a type defined by typedef (typedef name) is used in a function prototype declaration or a declaration using a const or volatile type modifier, the typedef expansion is illegal, and an error results.

[Description]

If a type defined by typedef (typedef name) is used in a function prototype declaration or a declaration using a const or volatile type modifier, the typedef expansion is illegal, and an error may result.

Example 1:

```
typedef int  FTYPE();

FTYPE  func;
int  func(void);                /* F713 Redefined 'func' */
```

Example 2:

```
typedef int  VTYPE[2];
typedef int  *VPTYPE[3];

const VTYPE  *a;
const int  (*a)[2];             /* F713 Redefined 'a' */
volatile VPTYPE  b[2];
volatile int *volatile  b[2][3];    /* F713 Redefined 'b' */
```

[Workaround]

There is no workaround.

[Correction]

Regard this item as a usage restriction.

No. 19 Sometimes a multidimensional array with an undefined size does not operate properly.

[Description]

Sometimes a multidimensional array with an undefined size does not operate properly.

Example 1:

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

Example 2:

```
char  c[][2][3]={"ab","cd","ef"};   /* Error (F756) */
```

[Workaround]

Define the size of the multidimensional array.

[Correction]

Regard this item as a usage restriction.

No. 20 In a function returning the address of a function with arguments, those arguments cannot be referenced. There is no error when referenced, but illegal code is output.

[Description]

In a function returning the address of a function with arguments, those arguments cannot be referenced. There is no error when referenced, but an illegal code is output.

Example:

```
char *c;
int  *i;
void (*f1(int *))(char *);
void (*f2(void))(char *);
void (*f3(int *))(void);

void main() {
        (*f1(i))(c);        /* Correct description (W510) */
        (*f1(i))(i);        /* Incorrect description */
        (*f2())(c);         /* Correct description (W509) */
        (*f2())();          /* Incorrect description (W509) */
        (*f3(i))();         /* Correct description (W509) */
        (*f3(i))(i);        /* Incorrect description */
}
```

W509 or W510 is output for a correct description. Nothing is output for a description that should produce a warning. However, the output code is normal.

```
void (*f4())(int p) {
       p++;                 /* Incorrect description */
}
```

An error is not output for a description that should cause an error. An illegal code is generated.

[Workaround]

There is no workaround.

[Correction]

Regard this item as a usage restriction.

No. 21 The signed type bit field is handled as an unsigned bit field.

[Description]

The signed type bit field is handled as an unsigned bit field.

[Workaround]

There is no workaround.

[Correction]

Regard this item as a usage restriction.

No. 22 If the total size of the auto variable in one function exceeds 65,535 bytes when the large model is used, the output code and debug information become illegal.

[Description]

If the total size of the auto variable in one function exceeds 65,535 bytes when the large model is used, the output code and debug information become illegal.

Example: when -ML is specified

```
void func(long a, int b){
     int i;
     char tab1[35000];
     char tab2[35000];
     i = b;
}
```

[Workaround]

Keep the total size of the auto variable in one function to within 65,535 bytes.

[Correction]

Regard this item as a usage restriction.

No. 23 An illegal code is output as a result of an operation that includes signed char type operator and constant values.

[Description]

An illegal code may be output if at least one of the conditions (1) to (5) is satisfied when the -QC1 option is specified, or if condition (6) is satisfied when the -QC1 or -QC2 option is specified.

<Conditions>

(1) If a right-shift operator “>>” is used, a left operand is a constant ranging from 128 to 255, and a signed char type operand is used on the right side.

(2) If a right-shift operator “/”, “%”, “<”, “<=”, “>”, “>=”, “/=”, or “%=” is used, either left or right operand is a constant ranging from 128 to 255, and a signed char type operand is used on the other side.

(3) If a binary operator operation is performed, in which either a left or right operand is a constant ranging from 128 to 255 and a signed char type operand is used on the opposite side, and obtained the result is converted into a type that is longer than 2-bytes.

(4) If a binary operator operation is performed, in which either a left or right operand is a constant ranging from 128 to 255 and a signed char type operand is used on the opposite side, and the obtained result is used as a left operand, for the right-shift operator “>>” with a signed char type operand as the right operand.

(5) If a binary operator operation is performed, in which either a left or right operand is a constant ranging from 128 to 255 and a signed char type operand is used on the opposite side, and the obtained result is used as an operand on a side, for the operator “/”, “%”, “<”, “<=”, “>”, “>=”, “/=”, or “%=” whose opposite-side operand is a signed char type operand.

(6) If a binary operator operation is performed, in which either a left or right operand is a constant that uses at least one of the operators “<<”, “>>”, “&”, “^”, or “|”, and the constant ranges from −128 to

255, the obtained result ranges from −128 to −1, and the type of the opposite-side operand is longer than 2-bytes.

Example 1:

```
signed char a;
if(a < 179/2){b++;}
```

Example 2:

```
int i, j;
i = j & (-127 | 4);
```

[Workaround]

Cast the relevant constant as a signed int type.

Example 1:

```
if(a < (signed int)179/2){b++;}
```

Example 2:

```
i = j & ((signed int)-127 | 4);;
```

[Correction]

This restriction will be corrected in Ver. 2.50.

A tool used to check whether this restriction applies or not is available.

Contact an NEC Electronics sales representative or distributor for details.

No. 24 An illegal code is output as a result of a compound assignment expression containing %= or /=.

[Description]

When all of the following three conditions are satisfied, a signed code is output where an unsigned division or remainder operation should be performed.

<Conditions>

(1) The expression includes a compound assignment operator “%=” or “/=”.

(2) The left side of the expression for (1) is the unsigned short type.

(3) The right side of the expression for (1) is the (signed) int type.
If it is a constant, it is not suffixed by u, U, l, or L, and ranges from −32768 to 32767.

Example:

```
unsigned short a;
a /= 0x5555;
```

[Workaround]

Use the simple assignment operator “=” instead of the compound assignment operators “%=” and “/=”.

Example:

```
a = a / 0x5555;
```

[Correction]

This restriction will be corrected in Ver. 2.50.

A tool used to check whether this restriction applies or not is available.

Contact an NEC Electronics sales representative or distributor for details.

No. 25 W503 is output when the array name of an automatic variable is referenced.

[Description]

W503 is output when the array name of an automatic variable without initialization is referenced.

W503 Possible use of '*variable-name*' before definition

Note:

"Initialization" means a declaration such as int a[2]={0,0};. It does not include assignment expressions such as a[0] = 0; a[1] = 0;.

An "array name" is 'a' in int a[2]. It does not include a[0], a[1], nor &a[0].

Example:

```
void func(void)
{
   int a[2];
   int *b;

   a[0] = 0;
   a[1] = 0;
   b = a;            /* W503 is output to this line */
}
```

[Workaround]

When W503 is output, check the relevant location. Ignore the message if initialization has been performed in the statement.

[Correction]

Regard this as a usage restriction.

3. Cautions

None.