To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. 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.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

===== Be sure to read this note. =====

C Compiler Package V.4.10 Release 1C
for the 79xx series
Release note
(Rev.5.0)

Renesas Solutions Corporation

Feb 16, 2006

Abstract

Welcome to C Compiler Package V.4.10 Release 1C for the 79xx series. This document contains supplementary descriptions to User's Manual. When you read certain items in the User's manual, please read this document as well.

1. Precautions on Product

When using the compiler, please be sure to follow the precautions and suggestions described below.

1.1. Precautions about C Compiler

1.1.1. On changing memory mapping in the startup program

In startup programs ncrt0.a79 and sect79.inc stored respectively in directory ¥src79¥startup and ¥smp79 under the directory where C Compiler Package has been installed, the interrupt vector table and the addresses of the interrupt processing function are for the 7902 group of MCUs. So, if C Compiler Package is used for microcomputers belonging to other groups than the 7902 group, for example, the 7910 or 7911 group, modify the interrupt vector table and the addresses of the interrupt processing function according to microcomputers involved.

- Modifications

(1) Interrupt Vector Tables

```
[ M37911/10 ]
                        .section    vector
                        .org        7fffb0H
        RESERVED15:     .word       OFFSET dummy_int
                          :
                          :
[ M37902/20 ]
                        .section    vector
                        .org        00ffc0H
        DMA3:           .word       OFFSET dummy_int
                          :
                          :
```

(2) Modified startup program sect79.inc for the 7910/7911 group:

```
[ M37911/10 ]
                        .section    interrupt
                        .org        7f0000H

[ M37902/20 ]
                        section     interrupt
                        .org        004000H
```

Note: 1. Assembler directive "OFFSET" must be added to the branch label in the interrupt vector table.

1.1.2. On arguments of formatted input/output functions

When the field width in the format specification of the following input/output functions is specified using the zero flag and an asterisk * not a decimal number but a character string is displayed:

- fprintf
- printf
- sprintf
- vfprintf
- vprintf
- vsprintf

For example, the flags in the printf("%0*d",keta,val) format specification denote the following:

```
0 ----- to right-align a decimal number and supply 0s enough to fill the number of digits to be displayed
* ----- to interpret argument "keta" as the number of digits to be displayed
d ----- to interpret argument "val" as a variable of type int and display it in a decimal number
```

However, an asterisk * that follows zero flag is not correctly interpreted to be a character in the format string, character string "*d" is displayed.

- Example

```
#include <stdio.h>

int         main(void)
{
            printf("%0*d¥n",4,123);
}
```

- Workaround

  Specify the field width in a decimal number.

```
#include <stdio.h>

int         main(void)
{
            printf("%04d¥n",123);
}
```

1.1.3. On standard library functions atof and strtod

If the argument of a standard library function atof or strtod is a character string beginning with a period (for example, ".12345"), the result of the conversion of the function becomes zero.

- Example

```
#include <stdlib.h>

double      d;

int         main( void )
{
            d = atof( ".12345" );      /* The character string of the argument begins with a period */
}
```

- Condition

  This problem occurs under the condition that if all the spaces contained in the argument are omitted, its character string begins with a period.

- Workaround

  Place a "0" in front of the period.

```
#include <stdlib.h>

double      d;

int         main( void )
{
            d = atof( "0.12345" );
}
```

1.1.4. On defining structure-type arrays

Writing an expression that references an element of a structure-type array may result in incorrect code being generated.

- Condition

  This problem occurs if the following four conditions are satisfied:

(1) A structure-type array is defined.
(2) A structure is defined.
(3) The definition in (1) precedes the one in (2).
(4) In the program exists an expression referencing an element of the array defined in (1).

- Example

```
struct AAA a[10];                    /* Condition(1) & Condition(3) */

struct AAA {                         /* Condition(2) */
        int     a;
        int     b;
};

int     gi;

void    smp(int i)
{
        gi = a[i].b;                 /* Condition(4) */
}
```

- Workaround

Define the structure in Condition (2) in advance of defining the structure-type array in Condition (1).

```
struct AAA {
        int     a;
        int     b;
};

struct AAA a[10];
```

1.1.5. On the standard library function "sprintf"

If a space is inserted between two arguments, % and f, of the "sprintf" standard library function, the result of an assignment may become such a value as 0.000000. (The number of decimal places varies according to the specified format of the sprintf function.)

- Condition

This problem occurs if a floating-point number explained below is assigned to argument "f".
This floating-point number is such a value as 0.9999999, which can be rounded off to 1 as the nearest whole number.

- Example

```
#include <stdio.h>

float   f;

int     main( void )
{
        char    buf[10];

        f = 0.999999;
        sprintf( buf,"% -8.2f",f );
}
```

- Workaround

This problem can be circumvented in either of the following ways:

(1) Modify the source file "print.c" of the standard library function as follows and re-create the standard library file by using the librarian:

```
if ( CHK_KETA ) {
        if ( (*format == 'e' || *format == 'E') && inte[0]=='9' ) {
                /*                              */
                inte[0] = '1';          /*                      */
                if ( CHK_EFUGO ) {
                        /*                      */
                        cnt--;
                        if ( !cnt )
                                /*                      */
                                CLR_EFUGO;
                } else
                        cnt++;
        } else {
                for ( r=0; r<seisu; r++ ) {
                        if ( inte[r] == '9' )
                                inte[r] = '0';
                        else {
                                ++inte[r];
                                break;
                        }
                }
                if ( r==seisu && r!=0) {
                        inte[seisu] = '1';
                        seisu++;
                }
                else if( seisu == 0){           // Processing to add
                        inte[seisu] = '1';      // Processing to add
                        seisu++;                // Processing to add
                }                               // Processing to add
        }
}
```

Re-create the standard library file by going through the following steps:

(1) Modify the print.c file saved in the SRCxx¥lib directory under the directory where your product is installed. Here xx denotes the numerals in each product type.

(2) Re-create the standard library file using the makefile.dos file saved in the SRCxx¥lib directory.

(3) Copy the re-created standard library file to the directory indicated by environment variable LIBxx.

(2) Pass the absolute value of a floating-point number to the sprintf function as its argument.

```
float       f;

int         main( void )
{
            char        buf[10];

            f = 0.9999999;

            /* The absolute value of a real number assigned after the first line of the buffer */
            /* No space inserted after % */

            buf[0] = ' ';                   /* A space assigned to buf[0] */
            sprintf( &buf[1],"%f",f );      /* The value of f assigned following buf[1] */
}
```

1.1.6. On Successively Referencing the Same Variable in More Than One if Statement

When two or more if statements contain the same variable, System Error may arise at compilation.

- Condition

  This problem may occur if the following six conditions are satisfied:

  (1) Any one or more of the -O, -O[1-5], -OR, and -OS compilation options are used.

(2) Within if-else constructs, if statements are nested in two or more levels at the else sides. (However, the innermost if statement is allowed to have no else statement.)
(3) The conditional expressions in all the nested if statements in (2) contain the same variable.
(4) After the if-else constructs in (2) exists an if statement whose conditional expression contains the same variable as described in (3).
(5) Before the if statement in (4) exists a program path that does not need to execute the if-else constructs in (2),
(6) Immediately before the if statement in (4) is placed an unconditional jump or a return.

- Example

(1) Example 1

```
int     a, b, cond;

void    func(void)
{
        if (a == 1) {                           /* Condition (5) */
                if (cond > 10) {                /* Conditions (2), (3): 1st nesting */
                        b += 1;
                } else if (cond > 5) {          /* Conditions (2), (3): 2nd nesting */
                        b += 2;
                } else if (cond > 3) {          /* Conditions (2), (3): 3rd nesting */
                        return;                 /* Condition (6) */
                }
        }
        if (cond == 1) {                        /* Condition (4) */
                b += 3;
        }
}
```

(2) Example 2

```
int     a, b, cond;

void    func(void)
{
        if (a == 1) {                                   /* Condition (5) */
                if (cond > 10) {                        /* Conditions (2), (3): 1st nesting */
                        b += 1;
                } else
                        if (cond > 5) {                 /* Conditions (2), (3): 2nd nesting */
                                b += 2;
                        } else {
                                if (cond > 3) {         /* Conditions (2), (3): 3rd nesting */
                                        b += 3;
                                }
                        }
                }
        } else {
                if (a == 2) {
                        return;                         /* Condition (6) */
                }
        }
        if (cond == 0x0001) {                           /* Condition (4) */
                b += 3;
        }
}
```

- Workaround

Place a dummy asm function immediately before the if statement in (4).

- Example 1 Modified

```
int         a, b, cond;

void        func(void)
{
            if (a == 1) {
                        if (cond > 10) {
                                    b += 1;
                        } else if (cond > 5) {
                                    b += 2;
                        } else if (cond > 3) {
                                    return;
                        }
            }
            asm();                          /* Place a dummy asm function */
            if (cond == 1) {
                        b += 3;
            }
}
```

1.1.7. On defining the data type of an array within a structure or union using a typedef statement

When the data type of an array within a structure or union is defined using a typedef statement, and then a variable is declared to be of the defined type with the near, far, or const qualifier being added, incorrect code may be generated or System Error may arise as follows:

- Condition

  This problem may occur if the following five conditions are satisfied:

  (1) A structure or union is defined.
  (2) The data type of an array within the structure or union in (1) is defined using a typedef statement.
  (3) A variable is declared to be of the type defined in (2).
  (4) The near, far, or const qualifier is added to the declaration in (3).
  (5) The structure or union in (1) is referenced.

- Example

```
struct tag {                                /* Condition (1) */
            long        l;
            char        c;
};

typedef   struct tag    ARR[3];             /* Condition (2) */
far   const   ARR       dat                 /* Conditions (3),and (4) */
                        = { 1, 2, 3, 4, 5, 6 };

void        func(int i)
{
            char c;

            c = dat[i].c + 1;               /* Condition (5) */
}
```

- Workaround

  Place a qualifier of the same type as used in (4) before the array in the typedef statement in (2).

```
struct tag {
        long    l;
        char    c;
};

typedef struct tag far      ARR[3];             /* Place another far before ARR */
far const ARR               dat = { 1, 2, 3, 4, 5, 6 };

void    func(int i)
{
        char c;

        c = dat[i].c + 1;
}
```

1.1.8. On using standard function libraries “scanf”, “fscanf”, and “sscanf”

In a scanf, fscanf, or sscanf function, conversion of an input character string that contains '0's by using the conversion specifier 'x' may not correctly be performed.

- Condition

  This problem occurs if the following four conditions are satisfied:

  (1) An input string contains a '0' or '0's.
  (2) A '0' in the string in (1) is converted using the conversion specifier 'x'.
  (3) The '0' in (2) is either of the following:
    (1) The '0' in (2) is at head of the input character string.
    (2) The character placed immediately before the '0' in (2) is not any of '0'--'9', 'a'--'f', and 'A'--'F'.
  (4) The character placed immediately before the '0' in (2) is not any of '0'--'9', 'a'--'f', and 'A'--'F'.

- Example
  - C source program

```
#include <stdio.h>
void    main(void);
void    func(void);

void    main(void)
{
        func();
}

void    func(void)
{
        int         input = 1234;
        int         returnVal = 0;
        const char  * pStr = "0";                   // Conditions (1), (3)(a), and (4)

        returnVal = sscanf(pStr, "%x", &input);     // Condition (2)
}
```

  - Execution results of the above program

```
returnVal = -1
input = 1234
        The correct results are returnVal = 1 and input = 0.
```

- Workaround

  Modify the several lines beginning at line 318 in the scan.c file saved in the src79¥lib subdirectory under the C Compiler Package -installed directory as follows:

  (1) Original

```
hex:
        data = 0L;
        if ( !width || !isxdigit( c ) )
                break;
        if(c=='0'){
                c=(*f_in)();
                if(c=='x'||c=='X'){
                        if(!isxdigit(c=(*f_in)()))
                                goto next;
                }
                else if(!isxdigit(c))
                        break;
        }
```

(2) Modified

```
hex:
        data = 0L;
        if ( !width || !isxdigit( c ) )
                break;
        if(c=='0'){
                c=(*f_in)();
                width--;                        // Add width--;
                if(c=='x'||c=='X'){
                        if(!isxdigit(c=*f_in)()))
                                goto next;
                }
                else if(!isxdigit(c)){          // Enclose break; with the
                        status = TRUE;          // curly braces { } and
                        break;                  // add status = TRUE; in front
                }                               // of break;.
        }
```

1.1.9. On jump addresses in switch statements

For switch statements having many jump addresses, the C compiler builds a jump table specifying all the jump addresses, places it in a sequence of execution instructions, and generates codes for indirect jumps by looking up this table.

However, compiling a C-language source file that contains a function in which two or more switch statements are described may lose part of the table to generate wrong codes, resulting in making incorrect jumps.

- Condition

  This problem may occur if the following six conditions are satisfied:

  (1) Option -OR is selected.

  (2) Option -ONBSD (-Ono_break_source_debug) is not selected.

  (3) Option -fST (-fswitch_table) is selected in C Compiler Package.

  (4) Two or more switch statements are described in a function.

  (5) After compilation, at least two switch statements are expanded each to a jump table and the codes that look it up to make indirect jumps.

  (6) The two switch statements in (5) meet either of the following conditions:

    (1) At any jump address of one switch statement exists an expression that is the same as the one residing at any jump address of the other.

    (2) Both switch statements have the path for breaking each statement without program execution in either of the following manners:

    (3) There is a case or default label that contains only a break statement for breaking the switch statement.

    (4) There is no default label.

- Example

```
extern int  a, b, c, x;

void        func(void)
{
            if (a) {
                        switch (b) {                    /* Condition (4): the 1st switch statement */
                        case 8:
                        case 9:
                        case 11:
                        case 12:
                        case 13:
                        case 14:
                        case 15:
                                    x++;                /* Condition(6) */
                                    break;
                        case 10:
                                    x = 2;
                                    break;
                        default:
                                    x = 0;               /* Condition(6) */
                                    break;
                        }
            } else {
                        switch (c) {                    /* Condition (4): the 2nd switch statement   */
                        case 33:
                        case 34:
                        case 35:
                        case 36:
                        case 37:
                        case 38:
                        case 39:
                        case 40:
                                    x++;                /* Condition (6); the same expression */
                                    break;
                        default:
                                    x = 0;              /* Condition (6); the same expression */
                                    break;
                        }
            }
}
```

- Workaround
  (1) For all the case and default labels that have the same expressions, add a dummy asm function immediately after each of them.
  (2) For all the case and default labels that have only a break statement, also add a dummy asm function in the same way.
  (3) For the switch statements with no default label, place a set of a default label, a dummy asm function, and a break statement in each switch statement.

  Note that when a statement contains several case and default labels, as cases 33 to 40 in the above example, not all the labels but only the last label requires a dummy asm function.

```
extern int  a, b, c, x;

void        func( void )
{
            if (a) {
                        switch ( b ) {
                        case 8:
                                    /* Omitted */
                        case 15:
                                    asm();              /* Dummy asm function */
                                    x++;
                                    break;
                        case 10:
                                    x = 2;
                                    break;
                        default:
                                    asm();              /* Dummy asm function */
                                    x = 0;
                                    break;
                        }
            } else {
                        switch (c) {
                        case 33:
                                    /* Omitted */
                        case 40:
                                    asm();              /* Dummy asm function */
                                    x++;
                                    break;
                        default:
                                    asm();              /* Dummy asm function */
                                    x = 0;
                                    break;
                        }
            }
}
```

1.1.10. Problem on Testing Bit Fields in an if Construct

When in the conditional expression of an if construct are included two (or more) expressions that are logically ANDed one another and each of which tests a bit field, incorrect code will be generated.

- Condition

  This problem occurs if the following six conditions are satisfied:

  (1) "-OR" compilation option is used along with "-O5" or "-O4".
  (2) Two (or more) expressions testing a bit field (stored in one bit) are included in the conditional expression of an if construct.
  (3) All the expressions in (2) test equality only (with operator "==").
  (4) More than one expression in (2) tests equality between a bit field and an immediate value of 1.
  (5) All the expressions in (2) are logically ANDed one another (with operator &&).
  (6) The program statement of the if construct is non or an unconditional jump.

- Example

```
sturct      bitf {
            int         b0:1;
            int         b1:1;
            int         b2:1;
}bit;
int         i;

void        func1(void)
{
            if((bit.b0 == 1)&&(bit.b2 == 1)){       /* Condition(2), (3), (4), (5) */
                        ;                           /* Condition(6) */
            }else{
                        i = 1;
            }
}

void        func2(void)
{
            if((bit.b0 == 1)&&(bit.b2 == 1)){       /* Condition(2), (3), (4), (5) */
                        goto L1;                    /* Condition(6) */
            }
            i = 1;
L1:;
}
```

- Workaround

  This problem will be circumvented in either of the following ways:

  (1) Use "-Ono_logical_or_combine" (-ONLOC) compilation option at compilation.

  (2) Describe a dummy asm function in the line immediately before the program statement of the if construct.

```
void        func1(void)
{
            if((bit.b0 == 1)&&(bit.b2 == 1)){
                        asm();                      /* A dummy asm function described */
            }else{
                        i = 1;
            }
}
```

1.1.11. On switch-case statements

When compiling switch-case statements, code that makes a jump only to the default or a specific case label may be generated no matter what case value meets the conditional expression.

- Condition

  This problem occurs if condition (1) or (2) below is satisfied with compile option -fswitch_table[-fST] selected.

  (1) In the conditional expression of a switch statement, a variable of type unsigned char is used, and the sequence of the case values is any of the following three types:
    (1) The case values are 255 consecutive numbers from 1 to 255.
    (2) The case values are 255 consecutive numbers from 0 to 254.
    (3) The total number of cases (excluding the default) is 143 or more with the minimum case value 0 and the maximum 255.

  (2) In the conditional expression of a switch statement, a variable of type signed char is used, and the sequence of the case values is any of the following three types:
    (1) The case values are 255 consecutive numbers from -127 to +127.
    (2) The case values are 255 consecutive numbers from -128 to +126.
    (3) The total number of cases (excluding the default) is 143 or more with the minimum case value -128 and the maximum 127.

- Example

```
char        c;

/* The case values are 255 consecutive numbers from 1 to 255.   */
switch( c ){
case   0:
case   1:
case   2:
            func( 3 );
            break;
              :
              :
case   255:
            func( 255 );
            break;
default:
            break;
}
```

- Workaround

  In the conditional expression of the switch statement, when the variable is of type unsigned char, cast it to an unisigned int value; when it is of type signed char, cast it to a signed int value.

```
char        c;

switch( (unsigned int )c ){          /* Variable c of type unsigned char is cast to of type unsigned int */
case   0:
              :
              :
case   255:
              :
              :
}
```

1.1.12. On Domain Errors Arising at Calling the "pow" Function out of the Standard Library

When a "pow" function is called out of the standard library, a domain error may arise even if values inside the domain are passed as arguments to the function.

- Condition

  This problem occurs if either of the following conditions is satisfied:

  (1) The first argument is zero and the second is positive.

  (2) The first argument is non-zero and the second is negative.

- Example

```
#include <math.h>
#include <error.h>

double      ans1;

int         func( void )
{
            ans1 = pow( 0.0 , 5.0 );          /* Condition(1)*/
            if( errorno == EDOM )
                        return   -1;          /* A domain error evaluated */
            return      0;
}
```

- Workaround

  Please modify the library source file included with your product, re-create the library, and re-link the user

programs.
For details of how to re-create libraries, refer to Article c "Incorporating the Modified Source Program" in Section E.3.2 "Sequence of Modifying I/O Functions" in your User's Manual.

- Modified statement

```
if(( x == 0    && y <= 0 ) | | ( x < 0 && y != (int)y )){
```

- Original statement

```
if( x == 0    | | y <= 0 ) | | ( x < 0 && y != (int)y )){
```

1.1.13. On Incorrect Optimization Made in a Loop Containing a Switch Statement

When a switch statement is described within a loop, optimization may be incorrectly performed at compilation, which moves the whole or part of a series of instructions not to be moved (see Note) into the line immediately before the loop.

Note: A series of those instructions that describe the operation never performed during the iterative execution of the loop

- Condition

This problem may occur if the following six conditions are satisfied:

(1) "-OS" compilation option is used at compilation.
(2) "-fswtich_table [-fST]" compilation option is selected.
(3) A switch statement exists within a for or while statement.
(4) Any of an auto variable, a register variable, and an argument has been assigned a new value at a case or default label to which the program branches.
(5) The value stored in the variable or argument in (4) does not change by iterative execution of a loop.
(6) Compilation generates a series of instructions for indirect branches represented by a branch destination table (see Note) from the switch statement.

Note:
A branch destination table is a list of labels representing branch destinations in assembler language corresponding to the case and default labels in a switch statement and is generated immediately after the indirect instructions that reference the table.

If the compiler judges it is efficient to indirectly branch by referencing the branch destination table, it will generate code for doing so. Although whether this judgment is made or not depends on the total number of case labels, the range of their values, and the continuity of the values, these conditions vary with product types.

- Example

```
int         func( void )
{
            int         j, a, flag = 0;         /* The first half of Condition (4): Here,  fig declared to be
                                                   an auto variable*/

            for( j = 0; j < 1; j++){
                        switch( a ){            /* Condition(3) */
                        case  8:
                                    break;
                        case  9:
                        case  10:
                        case  11:
                        case  12:
                        case  13:
                        case  14:
                        case  15:
                        default:
                                    flg = 1;    /* Conditions (4) and (5): A series of
                                                   instructions corresponding
                                                   to this expression moved incorrectly */
                                    break;
                        }
            }
            return  flg;
}
```

- Workaround

  Place a dummy asm function in front of the iterative processing where this symptom occurs.

```
            for( j = 0; j < 1; j++){
                        asm();                  /* Dummy asm(); placed */
                        switch( a ){
                        case  8:
                                    break;
```

1.1.14. On Forced Termination of Compilation by Describing Update of a Pointer Variable within a Loop

When into the value indirectly referenced by a pointer variable is described a processing that updates the pointer itself, compilation may forcefully be terminated. In such a case, an error message saying "This program has performed an illegal operation. . . ." will be displayed.

- Condition

  This problem may occur if the following four conditions are satisfied:

  (1) This problem may occur if the following four conditions are satisfied:
  (2) The value indirectly referenced by a pointer variable is written into the pointer itself.
  (3) The pointer variable in (1) is assigned to a register.
  (4) In the program exists a for or while statement that uses the pointer described in (1).
  (5) In the for or while statement in (3) resides an invariant expression that could be moved to the outside of the loop if the value in (1) were not indirectly referenced.

  If all the conditions above are met, however, the problem might not arise. In such a case, the code generated is good for use.

- Example

```
struct   l{
             unsigned int          no;
             struct   l    *next;
} *pp;

int          func( void )
{
             register struct   l       *p = pp;                    /* Condition(2) */
             int                       j;
             int                       n = 1;
             int                       x = 0;

             for(   j = 0; j < n-1; j++ ){                         /* Condition(4)   "n-1" is an invariant */
                          if( p->no > (p->next)->no )      x++;    /* Condition(3) */
                          p = p->next;                             /* Condition(1) */
             }
}
```

- Workaround

  Place a dummy asm function in front of the iterative processing where this symptom occurs.

```
             for(   j = 0; j < n-1; j++ ){
                          asm();                                   /* Dummy asm(); placed */
                          if( p->no > (p->next)->no )      x++;
             }
```

1.1.15. On Setting Initial Values for Arrays of Integer Types

Initial values for arrays of integer types may be compiled incorrectly.

- Condition

  This problem occurs if the following four conditions are satisfied
  (though it might not arise depending on constructions of C-language source programs even if all the conditions are met):

  (1) Initial values for an array of integer types are set at the same time the array is defined as global variables or static variables.
  (2) The initial value of the first element of the array in (1) is not an expression including an operator. (See Note.)
  (3) Any of the initial values of the second and later elements is an expression including an operator. (See Note.)
  (4) The value immediately after the expression including an operator in (3) is an integer constant greater than or equal to 256 and less than or equal to 342 in decimal notation.

  Note: A minus sign, a cast operator, and others are included.

- Example
  - Example 1:

```
int          array1[] = { 1,-5,302 };          /* NG */
int          array2[] = { -1,-5,302 };         /* OK */
```

  - Example 2:

```
int          j;
unsigned   long        addr1[] = { 0, (unsigned    long)& j; 208 };          /* NG */
unsgined   long        addr2[] = { (unsigned long)0 ; (unsigned    long)& j; 308 };  /* OK */
```

- Workaround

  If this problem occurs, place a cast operator in front of the initial value of the first element of the array.

1.1.16. On declaring type "enum"

Declaration of type "enum" with no tag name causes an error to appear which says "This program has performed an illegal operation . . . ." in Windows.

- Condition

  This problem occurs if the following two conditions are satisfied:

  (1) A pointer to type "enum" or an array of type "enum" with no tag name is declared.

  (2) Either of the following is performed using the pointer or array described in (1):

  (1) an assignment or comparison operation

  (2) a prototype declaration

- Example

```
enum { A_1, A_2 }       *ap;                    /* Condition(1) */
enum { B_1A_2 }         *bp;                    /* Condition(1) */

ap = bp;                                        /* Assignment operation: condition (2) */
ap < bp;                                        /* Comparison operation: condition (2)*/
ap != bp;                                       /* Comparison operation: condition (2) */

void        f( enum { C_1, C_2 } *);/* Prototype declaration: Condition(2) */

void        func( void )
{
            f( ap );
}
```

- Workaround

  Make sure to declare type "enum" with a tag name.

```
enum   e_a   { A_1, A_2 }       *ap;        /* Tag name "e_a" described */
enum   e_b   { B_1A_2 }         *bp;        /* Tag name "e_b" described */
```

1.1.17. On the Function-Extending #pragma STRUCT Directive

If "#pragma STRUCT tag name unpack" is described in front of an array of structures, no data designated as being unpack can be accessed properly.

- Condition

  This problem occurs if the following two conditions are satisfied:

  (1) An array of structures including members of type char is declared.

  (2) "#pragma STRUCT tag name unpack" is described in front of an array of structures in (1) above.

- Example

```
#pragma   STRUCT   S          unpack;           /* Condition(2) */

struct   S{
                   char       c;                /* Condition(1) */
                   int        j;
};
sturct   S   s[ 3 ] = { { 1,2 },{ 3,4 },{ 5,6 }};
```

- Workaround

  Do not use "#pragma STRUCT tag name unpack" for an array of structures including members of type char.

1.1.18. On #pragma ADDRESS

In V.3.20 Release 1, the address of the variable specified with the “#pragma ADDRESS” was treated as an absolute address. Therefore, change variable address values that are written as relative values from the bank value to absolute address.

1.1.19. On inline Function

If branch commands are used in the inline function, a system error may occur. If this happens, either remove the branch command from the inline function or use an ordinary function, not the inline function.

1.1.20. About the search of an include file

If you give a file to include together with a drive name in the #include line, and attempt to compile the file from a directory different from the one in which the file to compile is present, instances may occur in which the file to include cannot be searched.

1.1.21. To be taken when using #pragma ASM/ENDASM and asm()

- Regarding debug information when using #pragma ASM outside functions, if you write #pragma ASM anywhere outside functions, no C source line information will be output. For this reason, information regarding descriptions in #pragma ASM to #pragma ENDASM, such as error message lines when assembling or linking and line information when debugging, may not be output normally.
- C compilers generate code of arguments to be passed via registers and of register variables by analyzing their scopes. However, if manipulations of register values are described using inline assemble functions (such as #pragma ASM/#pragma ENDASM directives and asm function), C compilers cannot hold information on the scopes of the above-mentioned arguments and register variables. So, be sure to save and recover register contents on and from the stack when registers are loaded using inline assemble functions described above.

1.1.22. On regarding the preprocessing directive #define

To define a macro which will be made the same value as the macro ULONG_MAX, always be sure to add the prefix UL.

1.2. Precautions about Assembler

1.2.1. On linking relocatable module files with no sections

Successively linking relocatable module files with no sections may end in failure with an error message "value is undefined" or "Illegal format" displayed.

- Conditions

  This problem occurs if the following three conditions are satisfied:

  (1) Sixty-five or more relocatable files exist in the program.

  (2) Thirty two or more relocatable files with no sections (for example, those containing only the declarations of external variable and functions) are linked at the thirty-third and later in linking order.

  (3) Relocatable files with sections are linked after the files in (2) above.

- Example

```
ln79 @cmdfile

[ Linking Order     .r79 file ]
-----------------------------
 1                  file1.r79
       .....................
32                  file32.r79
-----------------------------
33                  file33.r79          32 relocatable files with no sections
       .....................
64                  file64.r79
-----------------------------
65                  file65.r79
-----------------------------
```

- Workaround

  This problem can be circumvented in either of the following ways:

  (1) Change the linking order so that 32 or more relocatable files with no sections cannot be in series.

  (2) Create and add relocatable files with zero-size sections for the same purpose as in (1).

  - Examples of source files with zero-size sections

```
- In C language
-------------------------------
#pragma ASM
.section empty
#pragma ENDASM
-------------------------------

- In assembly language
-------------------------------
.section empty
.end
-------------------------------
```

  - Modified example

```
[ Linking Order     .r79 file ]
-----------------------------
1                   file1.r79
       .....................
32                  file32.r79
-----------------------------
33                  file33.r79
34                  empty.r79           Create this relocatable file with zero-size sections and place it at the 34th
line.
       .....................
64                  file63.r79
-----------------------------
65                  file64.r79
66                  file65.r79
-----------------------------
```

1.2.2. Function-extending directive command “@“

When a character string containing an @ as a character constant is described in the operand of a .BYTE instruction, the @ may be interpreted as a concatenation operator.

- Condition

  This problem occurs if the following two conditions are satisfied:

  (1) Two or more character strings are described in the operand of a .BYTE instruction.

  (2) In the character strings in (1) above, one that contains a double quote " is described before another that

contains a character constant @.

- Example

```
.byte    "", "A@B"          ; Treated as "" , "AB" since the @ is interpreted as a concatenation operator.
```

Note that the description shown in the following example is treated properly because it does not satisfy Condition (2) above.

```
.byte    "A@B", ""          ; Treated as "A@B" , "" since the @ is interpreted as a character constant.
```

- Workaround

When a character string containing a double quote is described in advance of another containing a character constant @, split them into two lines or more.

```
.byte    ""
.byte    "A@B"                      ; Treated as it is.
```

1.2.3. The tag jump function

When TM (Integrated Development Environment) is used, the editor's tag jump function may not be carried out for error or warning messages sent by the assembler.

- Condition

This problem occurs if an error or warning message is displayed in the same line that a macro processing status message "---*---" is described in.

- Example

```
7900 Series Assembler system Version 4.10 Release1
Copyright 2000, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

( test.a79 )
macro processing now
----*----*test.a79 26 Warning (mac79): Actual macro parameters are not enough
test.a79 29 Warning (mac79): Actual macro parameters are not enough
                :
                :
```

Here, the tag jump function cannot be carried out for "test.a79 26 Warning (mac79): Actual macro . . ."

- Workaround

This problem will be circumvented in either of the following ways:

(1) Disable the macro processing status message "---*---" from being displayed on screen by using "-." (Even if you do this, error or warning messages can be displayed.) Assembly Option.

(2) Create a tag file for assembler errors with "-T" Assembly Option selected, and use an editor's tag jump function using this tag file.

1.2.4. On Directive Command ".ORG"

Describing more than one ".ORG" directive command in one CODE (program) section may assign more than one instruction to the same address, resulting in incorrect operations being performed.

```
            .section    prg, code
            .org        4000H
            bbs         #1122h, extsym1, lab
            movm        extsym2, #3000H
                        :
                        :
            movm        extsym2, #4000H
lab:
            .org        4100H
            lda.W       A, #5000H
                        :
```

- Condition

  This problem occurs if the following three conditions are satisfied:

  (1) More than one ".ORG" directive command are described in one CODE (program) section.

  (2) Jump instructions or subroutine-call instructions are described after the description of an ".ORG" directive command.

  (3) The location address (LOC.) of an object code generated by assembling is the same as the address specified by the second ".ORG" directive command or later.

- Example

```
SEQ.  LOC.    OBJ.                0XMSJA  ....*....SOURCE STATEMENT....

  3                                         .section  prg, code
  4   004000                                .org      4000H              --> Condition 1
  5   004000  414E0000r221102       J       bbs       #1122H, extsym1, lab --> Condition 2
              2003A7F600            J
  6   00400C  9600300000r                   movm      extsym2, #3000H
        :
        :
 36
 37   0040FD  9600400000r                   movm      extsym2, #4000H     --> Condition3
 38   004102                        lab:
 39
 40   004100                                .org      4100H              --> Condition 1
 41   004100  160050                        lda.W     A, #5000H
```

In the above example, two instructions described in lines 55 and 60 are assigned to address 0F0100H and address 0F0101H each.

- Workaround

When describing more than one ".ORG" directive command in one CODE (program) section, define another section name using directive command ".SECTION" and then describe ".ORG" directive commands.

```
            .section    prg, code
            .org        4000H
            bbs         #1122h, extsym1, lab
            movm        extsym2, #3000H
                        :
                        :
            movm        extsym2, #4000H
lab:
            .section    prg_1,CODE      ; Definition of another section by directive command
                                        ; ".SECTION" added.
            .org        4100H
            lda.W       A, #5000H
                        :
```

1.2.5. On using macro directive command ".LOCAL"

When a ".LOCAL" macro directive command and a character string are described in a macro definition, the character string may not correctly be expanded.

- Condition

  This problem occurs if the following three conditions are satisfied:

  (1) A macro local label is declared using macro directive command ".LOCAL" in a macro definition.
  (2) A character string is described in the macro definition in (1).
  (3) The character string in (2) ends with a period (".").

- Example

[Assembly-language source file]

```
mac     .macro
        .local      btop, bend          ; Condition(1)
        btop:
        .byte       bend - btop
        .byte       "string."           ; Condition(2), (3)
bend:
        .endm

        .section    prg,code
        mac
        .end
```

[Results of macro expansion]

```
        .local      btop, bend
..ml0001:
        .byte       ..ml0002 - ..ml0001
        .byte       "stringstring"      <— Expanded incorrectly
..ml0002:
        .endm
```

- Workaround

  Separate special character "." from the preceding character string and define the special character as an immediate value.

```
mac     .macro
        .local      btop, bend
btop:
        .byte       bend - btop
        .byte       "string", 2EH       ; Special character "." is defined as an immediate value.
bend:
        .endm
```

1.2.6. Concatenating strings in assemblers

When more than one argument of a macro call is concatenated, the line numbers contained in the line information after the concatenation get different from those in the source program. As a result, the following symptoms appear:

(1) If an error arises, the number of a line different from the one at which the error is detected will be displayed in the error message, which will prevent you from moving to the source line in question when the tag jump function is used in an editor and others.
(2) Source level debug will not properly be performed in simulator debuggers and debuggers.

- Example
  - test.a79

```
[line no]   [source]
 2          macTEST  .macro     _a, _b, _c
 3                   .byte      _a
 4                   .byte      _b
 5                   .byte      _c
 6                   .endm
 7
 8          macTEST  1, ¥¥                ; Arguments of a macro call concatenated
 9                   2, ¥¥
10                   3
11Ayte      err                           ; An error arises at this line
```

- Message

```
test.a79   9   Error(asp79): Undefined symbol exist ‘err’
```

- Workaround

  Refrain from concatenating arguments of macro calls.

1.2.7. On Directive Command ".INCLUDE"

When a source file is assembled, the error saying "Can't open include file" may arise.

- Conditions

  This problem occurs if the following two conditions are satisfied:

  (1) The name of an include file specified by a relative pathname is assigned to the operand of an ".INCLUDE" directive command.

  (2) The source file resides in a directory different from the current directory or the one where the include file has been saved.

- Example
  - Source file

```
include     inc¥b.inc                   ; Condition(1)
```

  - Directory tree structure and files

```
¥project -----   ¥work
          |
          ----- ¥src ----- a.a79        ; Condition (2)
                    |
                    ------ ¥inc ----- ¥b.inc   ; Condition (2)
```

  - Assembling (¥project¥work assumed to be the current directory)

```
C:¥> as79 ..¥src¥a.79
                        :
      Can’t open include file           ; An error arises.
```

- Workaround

  Place a ".¥" in front of the operand of an ".INCLUDE" directive command as ".¥inc¥b.inc".

1.2.8. On Location symbol “$“

When the source file where location symbol "$" is described in the operand of an unconditional branch instruction (BRA) is assembled, Assembler is forced to terminate.

- Example

```
.section    program
bra         $
.end
```

- Workaround

  When describing location symbol "$" in the operand of an unconditional branch instruction, use BRAL, JMP, or JMPL as the instruction.

1.2.9. On Using "-O" Assembly Option in the Load-Module Converter

When "-O" Assembly Option is used to specify the name of an output file, the file may be generated without its filename extension if a period (.) is contained in the filename (including its path).

- Example

```
> lmc79 -o ..¥output¥sample sample.x79
```

  In this example, the output file will be generated with the name "sample", not the correct name "sample.mot".

- Workaround

  If you specify an output file with a period (.) being contained in the filename (including its path), append filename extension ".mot" or ".hex" to the filename.

```
> lmc79 -o ..¥output¥sample.mot sample.x79          ; .mot is appended.
```

1.3. Precaution of MCU-Dependent Code

1.3.1. About 79xx Series-Dependent Code

(1) Make sure that the JSR, JMP, and RTS instructions are not mapped to the highest address in the bank or so that they cross the bank boundary. If there is a risk of this happening, specify the "-C" compilation option when linking. This option causes a warning message to be displayed if JSR, JMP, and RTS are mapped to a bank boundary.

(2) You may need to use specific instruction when to or reading registers in the SFR area. Because the specific instruction is different for each model, see the User's Manual for the specific Machine. These instructions should be used in your program using the asm function.

1.3.2. About access of SFR area

You may need to use specific instructions when writing to or reading registers in the SFR area. Because the specific instruction is different for each model, see the User's Manual for the specific Machine. These instructions should be used in your program using the asm function.

1.4. Precautions about TM

- As for integrated development environment TM, use Version 3.00 or a later version. C Compiler Package in this version cannot be used in Version 2.01 or in an earlier version. So be careful.
- In an attempt to divert a project generated by TM V.2 for TM V.3, the -finfo compilation option is not turned effective either in compiling or in assembling. Choose -finfo separately. For details, see the Release note of TM V.3.

1.5. Precautions about MS-Windows

1.5.1. Precautions about environment of operation

(1) C Compiler Package operates under Windows 95, Windows 98, Windows NT 4.0 or later. It does not work under Windows 3.1 and Windows NT 3.5x or earlier.

(2) If in Windows NT environment the command prompt size is set to other than "80 x 25," the command prompt size will change frequently as you start the compiler. Make sure the command prompt size is set to "80 x 25."

1.5.2. Suggetions Concerning File Names

The file names that can be specified are subject to the following restrictions:

- Directory and file names that contain kanji cannot be used.
- Only one period (.) can be used in a file name.
- Network path names cannot be used. Assign the path to a drive name.
- Keyboard shortcuts cannot be used.
- Directory and file names that contain a space character cannot be used.
- The "..." symbol cannot be used as a means of specifying two or more directories.
- A file name in length of 128 characters or more including path specification cannot be used.

1.5.3. Precautions about virus check programs

If the virus check program is memory-resident in your computer, C Compiler Package may not start up normally. In such a case, remove the virus check program from memory before you start C Compiler Package.

1.5.4. Precautions when upgrading

To upgrade C Compiler Package, uninstall the currently installed C Compiler Package first before you install the new version.

- Procedure for uninstalling C Compiler Package

  To uninstall C Compiler Package, launch Add/Remove Programs in Control Panel and then execute Uninstall.

2. Installing C Compiler Package

2.1. Before installing C Compiler Package

Please confirm as follows before installing C Compiler Package in your computer.

- Please carefully read the "License Agreement" and “Release Note” included with your product before using C Compiler Package. If you've installed this product in your computer, it is assumed that you've agreed to the provisions stipulated in the License Agreement.
- In order that C Compiler Package operates comfortably, it requires at least 32Mbytes of memory and a hard disk having 20Mbytes or more of space.
- Use the dedicated installer to install C Compiler Package.
- You need to input a license ID in the middle of installation. Before you start installing C Compiler Package, check your license ID.

2.2. C Compiler Package Installer

The installer is provided for each of the environments listed below. Check the product you've purchased to find the appropriate installer.

- Japnese environment

Supported host	Supported OS	Installer name	Directory on CD-ROM
IBM[1] PC/AT compatible	Microsoft Windows[2] 98 Microsoft Windows Me Microsoft Windows NT Microsoft Windows 2000 Microsoft Windows XP	SETUP.EXE	¥NC308WA¥W95J

- English environment

Supported host	Supported OS	Installer name	Directory on CD-ROM
IBM PC/AT compatible	Microsoft Windows 98 Microsoft Windows Me Microsoft Windows NT Microsoft Windows 2000 Microsoft Windows XP	SETUP.EXE	¥NC308WA¥W95E

2.3. Installation procedure

Please install C Compiler Package in the following procedure.

(1) Go to the directory corresponding to your system, which can be found the name of the software you purchased, on the CD-ROM.

(2) Start up the installer and follow the messages displayed on the screen as you install C Compiler Package.

2.4. Setting environment after installation

After you finished installing C Compiler Package, set environment variables next.

2.4.1. Environment settings for the C Compiler Package

The environment variables marked by "Auto" in the tables below do not need to be set because the Windows installer automatically rewrites AUTOEXEC.BAT.

Environment variable	Example of setting
BIN79	Auto (SET BIN79=C:¥MTOOL¥BIN)
INC79	Auto (SET INC79=C:¥MTOOL¥INC79)
LIB79	Auto (SET LIB79=C:¥MTOOL¥LIB79)

[1] IBM and AT are registered trademarks of International Business Machines Corporation.
[2] Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the U.S. and other countries.

TMP79	Auto (SET TMP79=C:¥MTOOL¥TMP)
NCKIN	SET NCKIN=SJIS
NCKOUT	SET NCKOUT=SJIS
Command path	Auto (C:¥MTOOL¥BIN is added)

3. Entering user registration

To be eligible for upgrade information, technical support, and other services, you must be regis-tered as a user with Renesas Technology Corporation. Unless you are a registered user, the said services cannot be received.
Please register your name with Renesas Technology Corporation **within 30 days after purchase.**

3.1. User registration of C Compiler Package

When you've installed C Compiler Package, the following file is created.

¥ mtool ¥ support ¥ nc79wa ¥ regist.txt

When you've installed the PC version of C Compiler Package, the following file is created.
Cut all contents of the regist.txt file and paste them into a file, then send it to the electronic mail address given below.

regist_tool@renesas.com

4. Contents of upgrade

4.1. Functional addition of C Compiler

4.1.1. #pragma TBLJMPOFF

Disables the functions designated with this directive from generating table jump code even when “-fswitch_table[-fST]“ compilation option.

4.1.2. -dsource_in_list[-dSL] compilation option

The C source list in the output assembler soruce list is generated into an assembler list as a comment; a list file is generated.

4.1.3. Wno_used_argument[-WNUA] compilation option

Outputs a warning for unused arguments.

4.2. Functional change of C Compiler

When a table jump code generated with "-fswitch_table[-fST] " compilation option extends over a bank boundary, the function to make it error at linking has been added.

4.3. Problem correction of C Compiler

Improvements have been made to all of the following precaution that had been Informed to you by tool news:

- Describing the condition expression that tests bit fields in an "if" statement may result in incorrect code being generated.
- Utilities utl79 may unexpectedly terminate their processing.

4.4. Problem correction of Assembler

Improvements have been made to all of the following precaution that had been Informed to you by tool news:

- On the total size of ROM displayed in linkers ln79
- On concatenating strings in assemblers
- On the order of searching include files in assemblers

5. Software version list of C Compiler Package V.4.10 Release 1C

The following lists the software items and their versions include with C Compiler Package.

- nc79 V.1.41.00
- cpp79 V.4.30.00
- ccom79 V.4.10.00
- as79 V.4.10.01
- mac79 V.3.20.01
- pre79 V.3.00.01
- asp79 V. 4.10.00
- ln79 V. 4.10.00
- lb79 V. 1.00.02
- lmc79 V. 3.20.00
- xrf79 V. 1.00.10
- abs79 V. 3.00.05
- stk79 V.1.00.00
- utl79 V.1.00.05
- sc79 V.1.00.00
- Map Viewer V.2.00.01
- Stk Viewer V.1.00.01

6. Versions Useful for the Realtime OS for the 79xx series

The C Compiler Package compiler presented here supports the Real-time Operating System V.2.20 Release 1. When you are using the Real-time Operating System in combination with the C Compiler Package, please be sure to use the above compiler version.