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.

REJ10J0439-0300

Integrated C Compiler for 77xx Series

M3T–NC77WA V.5.20 Release 4B

Release Notes

(13th Edition)

Renesas Solutions Corporation

Feb. 16, 2006

Abstract

Welcome to M3T–NC77WA V.5.20 Release 4B . 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.

Contents

1 Inquiry

On April 1, 2003, Mitsubishi Electric Semiconductor Application Engineering Corporation, a member of the Mitsubishi Electric group, joined the new Renesas Technology group and changed its name to Renesas Solutions Corp. Please note the following changes:

- User Registration
  - Old: regist@tool.mesc.co.jp
    regist@tool.maec.co.jp
  - New: regist_tool@renesas.com
- Tool Technical Support
  - Old: support@tool.mesc.co.jp
    support@tool.maec.co.jp
  - New: csc@renesas.com
- Tool Homepage
  - Old: http://www.tool-spt.mesc.co.jp/
    http://www.tool-spt.maec.co.jp/
  - New: http://www.renesas.com/
- Company Name
  - Old: Mitsubishi Electric Semiconductor Software Corp.
    Mitsubishi Electric Semiconductor Systems Corp.
    Mitsubishi Electric Semiconductor Application Engineering Corp.
  - New: Renesas Solutions Corp.

1.1 The latest info

Please refer to the useful following sites:
http://www.renesas.com/

2 Software version list

The following lists the software items and their versions included with NC77WA V.5.20 Release 4B .

- nc77 V.4.01.00
- cpp77 V.4.01.00
- ccom77 V.2.04.02(NC_CORE Version 2.02.04)
- loop77 V.1.20.00
- s2ie V.1.33.00
- stk77 V.1.21.00
- rasm77 V.5.10.00
- pre77 V.5.00.00
- br77 V.1.00.00
- link77 V.2.10.00
- lib77 V.5.00.00
- crf77 V.2.10.10

- hextos2 V.2.00.00
- lst77 V.1.00.10
- xref V.1.10.00

3 Operating Environment

The table below lists the host computers and OS versions with which NC77 and RASM77 have been confirmed to be operation normally.

Host computer	OS version	CD-ROM directory
SPARC station[1]	Japanese Solaris[2]2.5	SOLARIS
HP9000/720[3]	HP-UX [3]	HP700
IBM[4]PC/AT and its compatible machines	Microsoft[5]Windows [5] 98	W95J, W95E
	Microsoft Windows Me	W95J, W95E
	Microsoft Windows NT Workstation 4.0	W95J, W95E
	Microsoft Windows NT Workstation 2000	W95J, W95E
	Microsoft Windows XP	W95J, W95E

Whether NC77 and RASM77 operate normally with other computers and OSs depends on manufacturers of those host computers and OSs. Therefore, please contact the manufacturers of your host computers and OS to see if the software will work properly with your computer system.

4 Installing NC77WA

4.1 Before installing NC77WA

Please confirm as follows before installing NC77WA in your computer.

- Please carefully read the ”License Agreement” and ”Release Notes” included with your product before using NC77WA. 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 NC77WA operates comfortably, it requires at least 32 Mbytes of memory and a hard disk having 20 Mbytes or more of space.
- Use the dedicated installer to install NC77WA.
- You need to input a license ID in the middle of installation. Before you start installing NC77WA, check your license ID.
- If your license ID is for the RASM77 product, you cannot install the NC77WA.

4.2 NC77WA Installer

Supported host	Supported OS	Installer name	Directory on CD-ROM
PC	Windows 95 Windows 98 Windows Me Windows NT Windows 2000 Windows XP	setup.exe	`\NC77WA\W95E`

[0] SPARC and SPARCstation are registered trademarks of SPARC International, Inc.
[1] Sun and SunOS are registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.
[2] Solaris is a registered trademark of Sun Microsystems, Inc.
[3] HP 9000 and HP-UX are trademarks of Hewlett-Packard Company of the U.S.
[4] IBM, AT, PS/2, and OS/2 are registered trademarks of International Business Machines Corporation.
[5] Microsoft, MS, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the U.S. and other countries.

4.3 Installation

Please use the following procedure for installation on a PC.

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 NC77WA or RASM77.

The data you input in the middle of installation is necessary to create a file for user registration. (Only the Windows installer creates registration files.)

4.4 Setting environment after installation

4.5 environment variables

After you finished installing NC77WA, set environment variables next.

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
BIN77	Auto (SET BIN77=C:\MTOOL\BIN)
INC77	Auto (SET INC77=C:\MTOOL\INC77)
LIB77	Auto (SET LIB77=C:\MTOOL\LIB77)
TMP77	Auto (SET TMP77=C:\MTOOL\TMP)
NCKIN	SET NCKIN=SJIS
NCKOUT	SET NCKOUT=SJIS
Command path	Auto (C:\MTOOL\BIN is added)

4.6 Environment value NCKIN and NCKOUT

- NCKIN

  This environment variable specifies the input code used. It can be selected from the following two:

  – EUC (Extended Unix Code)
  Japanese characters can be written. However, external code consisting of 3 bytes per character cannot be used.
  – SJIS
  Japanese characters can be written.

- NCKOUT This environment variable specifies the output code used. It can be selected from the following two:

  – EUC
  – SJIS

The default codes of NCKIN and NCKOUT (those automatically selected if you do not specify) are as follows:

Host name	NCKIN/NCKOUT	Default code
PC	NCKIN	SJIS
	NCKOUT	SJIS

When using the compiler in areas outside Japanese and English-speaking countries (e.g., European language area), be sure to choose EUC.

5 User registration

To be eligible for upgrade information, technical support, and other services, you must be registered 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.

5.1 Registering your name

1. When you've installed the PC version of NC77WA, the following file is created.

   ```
   \mtool\support\nc77wa\regist.txt
   ```

   mtool is the directory created by default when you installed NC77WA.

2. 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@renesas.com
   ```

If you are not using an electronic mail, output the contents of the regist.txt file to a printer and send the printout by facsimile. See the Release Notes included with your product to find the address to be sent.

6 Changes from NC77 V.5.10 Release1

6.1 Addition of Options

6.1.1 Optimization Options

The following new option has been added:

–Ono_logical_or_combine(–ONLOC)

- Suppresses the optimization that puts consecutive ORs together.

–Ocompare_byte_to_word(–OCBTW)

- Compares consecutive bytes of data at contiguous addresses in words.

6.1.2 Warning Options

The following new option has been added:

–Wuninitialize_variable(–WUV)

- Overview

  Outputs a warning for using un-initialized auto variables.

- Notes

  The line No. in the warning message is the last line in the function (normally the "}" line).

  And, when it is initialized by a conditional branch (generated by an "if", "for" etc.), the compiler assumes that it is not initialized. Therefore, a warning is issued.

```
example)
main()
{
        int i;
        int val;
        for(i = 0; i < 2; i++){
                f();
                val = 1; // Logically, always initialized here.
        }
        ff( val );
}
```

–Wlarge_to_small(–WLTS)

- Overview

  Outputs an warning for implicit transfers from large size to smaller size.

- Notes

  This option does not work even when the –Wall option has been specified. This is because numerous warnings can occur when this option is used. To use all warning features of the NC77, simultaneously specify –Wall and –WLTS.

6.2 Addition of #pragma MX1FUNCTION

1. Overview

   A #pragma MX1FUNCTION that operates like the #pragma M1FUNCTION has been added. It is called when both the M and X flags are “1”.

2. Description

   - Pragma name
     `#pragma MX1FUNCTION`
   - Function
     Calls the function and sets the default to “M flag = 1” and “X flag = 1” when defining the default.

6.3 Addition of “$@” in asm function

Added processing for asm function `$@` has been added. When `$@` has been specified, the compiler identifies and outputs whether the said variable is an auto variable, register variable or external variable.

6.4 Improvements Over NC77 V.5.10

The following cautions, detailed in Tool News, are no longer required:

TOOL NEWS	Item
May.16.1999 (MESCT–NC308–990516D)	Problem on Arrays of Type far with Initial Values
	Problem on ”sizeof” Operator
June.16.1999 (MESCT–NC77WA–990616D)	On Accesses to Arguments of Functions
	On IEEE-695 File Converter s2ie
Aug.1.1999 (MESCT–NC79WA_2–990801D)	standard library function ”longjmp”
Oct.16.1999 (MESCT–NC79WA–991016D)	An integer constant cast to a pointer

7 Changes from NC77 V.5.20 Release1

7.1 Improvements Over NC77 V.5.20 Release1

The following cautions, detailed in Tool News, are no longer required:

TOOL NEWS	Item
Jan.16.2000 (MESCT–NC308WA–000116D)	Initialization of external variables of type float or double may result in incorrect values being assigned.
Feb.1.2000 (MESCT–NC308WA_1–000201D)	When programs containing variables that are declared to be ”const” are compiled, system errors may arise.

8 Changes from NC77 V.5.20 Release2

8.1 Improvements Over NC77 V.5.20 Release2

The following cautions, detailed in Tool News, are no longer required:

TOOL NEWS	Item
April 1.2000 (MESCT–CC32R–000401D)	If a product is installed by root user, its user and group IDs both become indefinite, so the product may not be accessed after installation.
April 1.2000 (MESCT–NC77WA–000401D)	If two or more sections with the same name are defined successively in a file, an error occurs in LST77 at the definition of the succeeding section.
May 1.2000 (MESCT–NC308WA–000501D)	If an optimize option is used, Windows' error message "This program has performed an illegal operation..." may appear.
June 16.2000 (MESCT–RASM77–000616D)	At typing a command, selecting only an option without specifying the name of a file to convert will force termination of LST77.
July 16.2000 (MESCT–NC308_1–000716D)	Folding of floating-point constants may cause incorrect subtract operations to be performed.
July 16.2000 (MESCT–NC308_1–000716D)	Folding constants of type unsigned long may cause incorrect divide and modulus operations to be performed.

9 Changes from NC77 V.5.20 Release3

9.1 Improvements Over NC77 V.5.20 Release3

The following cautions, detailed in Tool News, are no longer required:

TOOL NEWS	Item
November 16.2000 (MESCT–NC77WA–001116D)	On codes created when bit fields are used both in "if" and "else if" statements in an "if - else if" construct

10 Changes from RASM77 V.5.00

10.1 Changed RASM77

It corrected the problem to develop the macro argument expansion which was described in the condition false part of the condition assembly.

10.2 Changed LINK77

1. Added command option –BRAL

   LINK77 outputs warning message when processing with this option if the code data which the jump address of BRAL is xxFFFFH is found.

   Confirm that the found data is under BRAL mnemonic, because LINK77 searches machine language file for the machine code pattern.

2. Added warning message

   ```
   WARNING NO.1: BRAL specified address is xxFFFFh (description address xxxxh)
   ```

   It is possible that BRAL mnemonic which jump address is xxFFFFh was located at xxxxh address.

10.3 Added BR77

It added branch optimizing tool br77 to the product. BR77 is the tool which changes the branch instruction which was made the "out of range" error into the optimal branch instruction.

11 Precautions

11.1 Installation

- About the installing directory name

  Can't specify the installing directory including space. Please refer to NC77WA/RASM77 Guide BOOK "Precautions on PC Version" about other precautions.

- When installing the Windows95 version

  When changing the item "Installation directory" from the "Select components" screen by using "Refer(R)" during installation, a trouble sometimes occurs that you cannot select some of the drives displayed by "Drive(V)" on the "Select directory" screen. In such a case, return to the "Select components" screen and use "Disk capacity(S)" on that screen to specify a drive. If you still have difficulty, re-execute the installer after restarting Windows95.

- When installing the HP9000/700 HP-UX10.X version

  On HP-UX, you may sometimes have a symptom that CD-ROM file name are shown in uppercase letters or something like ";1" is added to the displayed file names. In such a case, since product cannot be removed, re-mount the drive following the procedure described below.

  [CAUTION]

  - This procedure must be executed by the super-user.
  - Since the CD-ROM drive name /dev/dsk/c0t2d0 varies with each host computer, be sure to check the drive name of your CD-ROM.
  - The mount point(/cdrom) is just an example.
  - For details about the commands used in the example below, refer to user's manual of your HP-UX.
  - The symbol "%" means the prompt.

```
% pfs_mountd &
% pfsd 4 &
% pfs_mount -t iso9660 -x unix /dev/dsk/c0t2d0 /cdrom
```

11.2 Precautions to be taken when using NC77WA

11.2.1 The x flag may not indicate correct status.

1. Description

   In the processing that comes with backward jumps, the "for" statement for example, the x flag may not indicate correct status.

2. Conditions

   This problem occurs if the following four conditions are satisfied:

   (a) Data of type char and of other types co-exist within the range of backward jumps made with "goto" (backward jumps only), "for", "while", or "do-while" statements (hereafter called a backward jump block).

   (b) In the first statement of the backward jump block in (1), a code that compares a constant with the contents of the X or Y register is generated by the compiler. (In Example below, the contents of register X is used as the offset value of s.ch1.)

   (c) In the last statement of the backward jump block in (1), a code that compares a constant of 0 with the contents of register X or Y is generated. (In Example below, the contents of register X is used as the offset value of uc[i].)

   (d) The program jumps to the beginning of the backward jump block with the X flag being set or cleared depending on the evaluation of the conditional expression in (3).

3. Example

```
/* In this example, the problem occurs
                when compile option -OS used. */
struct S{
    unsigned char   ch1;
    unsigned char   ch2;
} s;

unsigned char   uc[5];

void func(void)
{
    int i, j;

    for( i=0; i<2; i++ ){                   /* Condition (1) */
        if( s.ch1 == 1 ){                   /* Condition (2) */
            uc[i] = 1;
        }

        if( (uc[i] == 5) || (uc[i] == 0) ){ /* Condition (3) */
            uc[i] = 0;
        }
        /* Condition (4)
         * If evaluation TRUE, program jumps with X flag set to 1;
         * otherwise, with X flag cleared to 0.
        */
    }
}
```

4. Workaround

   If this problem arises, force the value of the X flag to change by adding an asm function immediately after the last backward jump block.

```
struct S{
    unsigned char   ch1;
    unsigned char   ch2;
} s;

unsigned char   uc[5];

void func(void)
{
    int i, j;

    for( i=0; i<2; i++ ){
        if( s.ch1 == 1 ){
            uc[i] = 1;
        }

        if( (uc[i] == 5) || (uc[i] == 0) ){
            uc[i] = 0;
        }
        asm(2,0);  /* Added to clear the X flag only to 0 */
    }
}
```

11.2.2 On division and modulus operations

1. Description

   When performing division or modulus operations involving variables of type unsigned long, the run-time library for these operations is called out without clearing the X flag.

2. Conditions

   This problem occurs if the following four conditions are satisfied:

   (a) A dividend is a variable
   (b) A divisor is a constant
   (c) Either or both of the dividend and divisor are of type unsigned long
   (d) A subscript of an array that takes an argument of type char is used in the conditional expression in an iteration statement immediately before a division or modulus operation

   Note:

   If the above conditions are all met, no problem may occur depending on the constructions of C source programs. If so, use your C compiler as it is.

3. Example

```
void func(unsigned char ch, unsigned long *array)
{
    unsigned long tmp;

    while( ch-- != 0 ){        /* Conditions (4) */
        array[ch] = tmp / 10;  /* Conditions (1), (2), and (3), */
    }
}
```

4. Workaround

   If this problem arises, change the type of the subscript of the array from char to signed char or unsigned char.

```
void func(unsigned int ch, unsigned long *array)
{
    unsigned long tmp;

    while( ch-- != 0 ){        /* Conditions (4) */
        array[ch] = tmp / 10;  /* Conditions (1), (2), and (3), */
    }
}
```

11.2.3 Problem on Setting Initial Values for Arrays of Integer Types

1. Description

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

2. Conditions

   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):

   (a) Initial values for an array of integer types are set at the same time the array is defined as global variables or static variables.

   (b) The initial value of the first element of the array in (1) is not an expression including an operator. (See Note.)

   (c) Any of the initial values of the second and later elements is an expression including an operator. (See Note.)

   (d) 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.

3. Example

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

Example 2:
    int i;
    unsigned long addr1[] = {0,(unsigned long)&i,308};/* NG */
    unsigned long addr2[] = {(unsigned long)0,
                             (unsigned long)&i,308};  /* OK */
```

4. Workaround

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

```
Example 1:
    int array1[] = {(int)1,-5,302};    /* Cast operator "int" added */

Example 2:
    int i;
    unsigned long addr1[] = {(unsigned long)0,
                             (unsigned long)&i,308};
                             /* Cast operator "unsigned long" added */
```

11.2.4 On Domain Errors Arising at Calling the ”pow” Function out of the Standard Library

1. Description

   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.

2. Conditions

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

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

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

3. Example

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

double  ans1;

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

4. 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.

   The necessary modification is as follows:

   - Whereabouts

     The first if statement in the “pow” function in library source file “pow.c” (about the 5th line from the top of the statement)

   - Modified statement

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

   - Original statement

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

11.2.5 On Incorrect Optimization Made in a Loop Containing a Switch Statement

1. Description

   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

2. Conditions

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

   (a) Optimizing option "–OS" is used at compilation.
   (b) Option "-fST" is selected.
   (c) A switch statement exists within a for or while statement.
   (d) 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.
   (e) The value stored in the variable or argument in (4) does not change by iterative execution of a loop.
   (f) 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.

3. Example

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

    for (i = 0; i < 1; i++) {
        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;
}
```

4. Workaround

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

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

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

1. Description

   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.

2. Conditions

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

   (a) The value indirectly referenced by a pointer variable is written into the pointer itself.

   (b) The pointer variable in (1) is assigned to a register.

   (c) In the program exists a for or while statement that uses the pointer described in (1).

   (d) 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.

Note:

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

3. Example

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

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

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

4. Workaround

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

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

11.2.7 On switch-case statements

1. Description

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.

2. Conditions

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

(a) 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:

i. The case values are 255 consecutive numbers from 1 to 255.
ii. The case values are 255 consecutive numbers from 0 to 254.
iii. The total number of cases (excluding the default) is 143 or more with the minimum case value 0 and the maximum 255.

(b) 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:

i. The case values are 255 consecutive numbers from -127 to +127.
ii. The case values are 255 consecutive numbers from -128 to +126.

iii. The total number of cases (excluding the default) is 143 or more with the minimum case value -128 and the maximum 127.

3. Example

```
 In condition (1)-i:
-------------------------------------------------------------------------
    char c;
    switch(c){
    case    1:
    case    2:
            func(3);
            break;
            :
            :
    case    255:
            func(255);
            break;
    default:
            break;
    }
```

4. Workaround

In the conditional expression of the switch statement, when the variable is of type unsigned char, cast it to an unsigned 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
    case    1:                       is cast to of type unsigned int */
        :
        :
    case    255:
        :
    }
```

11.2.8 On testing equality between the value that a function returns and a constant of 0

1. Description

When a function returns a value of type signed or unsigned char, testing equality between this returned value and a constant of 0 will be performed in an incorrect way. The reason is that the instruction for changing the status of the M or X flag is not correctly generated at compilation, so the test is conducted in 16 bits wide, not 8 bits.

2. Conditions

This problem occurs if the following three conditions are satisfied:

(a) An equality test is performed between the value a function returns and a constant of 0.

Note that (1) conditional expressions in if, do, while, and for statements are involved, and (2) the type of description ”func( )” can also be tested with a constant of 0.

(b) The function returns a value of type signed or unsigned char.

(c) When calling the function, its arguments are saved on the stack.

Note that if all the arguments are passed to the function via registers, Condition (3) is not met since the stack is not involved.

3. Example

   When the following C-source program is compiled, the instruction for changing the status of the M flag (the sem instruction that would be generated immediately before the test instruction) is not generated, resulting in an incorrect test being performed.

```
unsigned char func(unsigned char, unsigned char); /* Condition (2) */
volatile int i;

void xxx( unsigned char c )
{
    if( func( c, 1 ) == 0 )                       /* Condition (1) */
        i = 0;
    else
        i = 1;
}
```

4. Workaround

   When the following C-source program is compiled, the instruction for changing the status of the M flag (the sem instruction that would be generated immediately before the test instruction) is not generated, resulting in an incorrect test being performed.

```
unsigned char func(unsigned char, unsigned char); /* Condition (2) */
volatile int i;

void xxx( unsigned char c )
{
    if( func( c, 1 ) == 0 )                       /* Condition (1) */
        i = 0;
    else
        i = 1;
}
```

11.2.9 On jump addresses in switch statements

1. Description

   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.

2. Conditions

   This problem occurs if the following six conditions are satisfied:

   (a) Option –OR is selected.

   (b) Option –ONBSD (–Ono_break_source_debug) is not selected.

   (c) Option –fST (–fswitch_table) is selected.

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

   (e) 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.

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

i. 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.

ii. Both switch statements have the path for breaking each statement without program execution in either of the following manners:

- There is a case or default label that contains only a break statement for breaking the switch statement.
- There is no default label.

3. Example

C-language source file:

```
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)-(i); #1 */
            break;
        case 10:
            x = 2;
            break;
        default:
            x = 0;      /* Condition (6)-(i); #2 */
            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)-(i);
                                the same expression as #1 above */
            break;
        default:
            x = 0;      /* Condition (6)-(i);
                                the same expression as #2 above */
            break;
        }
    }
}
```

4. Workaround

   (a) For all the case and default labels that have the same expressions, add a dummy asm function immediately after each of them.

   (b) For all the case and default labels that have only a break statement, also add a dummy asm function in the same way.

   (c) 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;
        }
    }
}
```

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

1. Description

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

2. Conditions

   This problem occurs if the following six conditions are satisfied:

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

   (b) 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.)

(c) The conditional expressions in all the nested if statements in (2) contain the same variable.

(d) After the if–else constructs in (2) exists an if statement whose conditional expression contains the same variable as described in (3).

(e) Before the if statement in (4) exists a program path that does not need to execute the if-else constructs in (2),

(f) Immediately before the if statement in (4) is placed an unconditional jump or a return.

3. Example

```
C-language source file: 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;
    }
}
```

```
C-language source file: 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;
    }

}
```

4. 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;
    }
}
```

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

1. Description

   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.

2. Conditions

   This problem occurs if the following four conditions are satisfied:

   (a) A structure or union is defined.

   (b) The data type of an array within the structure or union in (1) is defined using a typedef statement.

   (c) A variable is declared to be of the type defined in (2).

   (d) The structure or union in (1) is referenced.

3. Example

```
C-language source file:

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) */
}
```

4. 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;
}
```

11.2.12 On the standard library function “sprintf”

1. Description

   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.)

2. Conditions

   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.

3. Example

```
C-language source file:

#include <stdio.h>
float   f;
int     main( void )
{
    char   buf[10];
    f = 0.9999999;
    sprintf( buf,"% f",f );  /*A space inserted between % and f*/
                             /*In the above example, the value assigned to
                                                    "buf" becomes 0.000000.*/
}
```

4. Workaround

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

   - Modify the source file “print.c” of the standard library function as follows and re-create the standard library file by using the librarian:

     Modification of “print.c”:

     To the processing in the _f8prn function, add the four lines with the comment of “Processing to add” as shown in an example below.

     Note:

     This modification part begins at the 890th line in the M3T-NC77WA V.5.20 Release 4, and this beginning line varies according to compilers and their versions.

```
    if ( CHK_KETA ) {
        if ( (p_flg == 'e' || p_flg == 'E') && inte[0]=='9' ) {
        /* %e && integer = 9 */
            inte[0] = '1';          /* set '1' integer part */
            if ( CHK_EFUGO ) {
                /* exponent < 0 (minus) */
                cnt--;
                if ( !cnt )
                    /* exponent = 0 -> to PLUS */
                    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:

(a) Modify the print.c file saved in the SRC77
lib directory under the directory where your product is installed.

(b) Re–create the standard library file using the makefile.dos file saved in the SRC77
lib directory.

(c) Copy the re–created standard library file to the directory indicated by environment variable LIB77.

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

```
#include <stdio.h>
#include <math.h>       /* math.h included */
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] */
}
```

11.2.13 On standard library functions atof and strtod

1. Description

   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.

2. Conditions

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

3. Example

```
C-language source file:

#include <stdlib.h>
double   d;
int     main( void )
{
    d = atof( ".12345" ); /* The character string of
                             the argument begins with a period */
}
```

4. Workaround

   Place a "0" in front of the period.

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

11.3 Sample of startup program

For C-language programs to be "burned" into ROM, NC77 comes with a sample startup program written in the assembly language to initial set the hardware (77xx), locate sections, and set up interrupt vector address tables, etc. This startup program needs to be modified to suit the system in which it will be installed.

11.4 #pragma ASM

When #pragma ASM is used outside a function, no C source line information is output for that location to the assembly file.

For this reason, the line numbers of error messages may not be correctly output for the code between #pragma ASM and #pragma ENDASM during assembling or linking or line data, etc., during debugging.

11.5 #pragma ASM, asm() and branch instructions

With NC77, the content written in the #pragma ASM and asm functions is output directly in the assembly file. If complicated branches which include C language source are created with #pragma ASM such as that shown below, the area storing the C language variables cannot be correctly analyzed, hence the correct code cannot be generated.

```
int far flag;

int func(void)
{
        int     i;
        int     j;

        i = 1;
#pragma ASM
        lda     A, LG:_flag
        beq     LABEL2
LABEL1:
#pragma ENDASM
        return i;

#pragma ASM
LABEL2:
#pragma ENDASM
        j = 2;
#pragma ASM
        lda     A, LG:_flag
        cmp.W   A, #0001
        beq     LABEL1
#pragma ENDASM
        return j;
}
```

11.6 #pragma ASM, asm() and registers

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.

```
int   k;
int   func(int i)
{
    int        j;

    asm( "     pha" );         /* Saves register contents */
    asm( "     lda.W   A,#000AH" );
    asm( "     sta.W   A,DT:_k");
    asm( "     pla" );         /* Recovers register contents */

    j = i + 100;
    return  j;
}
```

11.7 Debugging information

Any changes have not been made to the debugging information the option –gie outputs. Thus the following restriction arises.

- You cannot reference register variables.

  Variables are assigned to a register only when you specify the register storage class and when you specify the option –Oenable_regsiter in a C source file. NC77's debugging information has nothing to do with register variables. Thus you cannot reference register variables.

  As for register arguments, you can reference them in the same manner as before, since they are laid in the stack after getting into a function.

11.8 Memory Management Functions

In addition to the user area, the heap area also includes a "management area used by memory management functions". Therefore, if you only reserve enough heap area for the user area, there is a risk that the heap area will be insufficient when the program is run.

11.9 –Ostack_align Option

The –Ostack_align option is effective only when the –O[1–5] optimization option has been specified. To use the –Ostack_align option, be sure to specify –O[1–5].

11.10 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.

11.11 Symbol Length Recognizable by s2ie

The s2ie can recognize symbol names up to 127 characters long. If symbol names of 128 characters or more are used, the s2ie cannot correctly generate the IEEE-695 file. Use symbol names of 127 or less characters.