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.

RENESAS

To all our customers

Regarding the change of names mentioned in the document, such as Mitsubishi Electric and Mitsubishi XX, to Renesas Technology Corp.

The semiconductor operations of Hitachi and Mitsubishi Electric were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Mitsubishi Electric, Mitsubishi Electric Corporation, Mitsubishi Semiconductors, and other Mitsubishi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Note : Mitsubishi Electric will continue the business operations of high frequency & optical devices and power devices.

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

User's Manual

NC77 V.5.20

User's Manual

C Compiler
for 77XX Series

Renesas Electronics
www.renesas.com

Rev.1.00 1999.11

Keep safety first in your circuit designs!

Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation put the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of non-flammable material or (iii) prevention against any malfunction or mishap.

Precautions to be taken when using this manual

- These materials are intended as a reference to assist our customers in the selection of the Mitsubishi semiconductor product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Mitsubishi Electric Corporation, Mitsubishi Electric Semiconductor Systems Corporation or a third party.
- Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation assume no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and is subject to change by Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Mitsubishi Electric Corporation, Mitsubishi Electric Semiconductor Systems Corporation or an authorized Mitsubishi Semiconductor product distributor for the latest product information before purchasing a product listed herein.

  The information described here may contain technical inaccuracies or typographical errors. Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation assume no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.

  Please pay attention to information published by Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation by various means, including Mitsubishi Semiconductor Homepage (http://www.mitsubishichips.com/) and Mitsubishi Tool Homepage (http://www.tool-spt.mesc.co.jp/index_e.htm).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation assume no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Mitsubishi Electric Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Mitsubishi Electric Corporation, Mitsubishi Electric Semiconductor Systems Corporation, or an authorized Mitsubishi Semiconductor product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Mitsubishi Electric Corporation or Mitsubishi Electric Semiconductor Systems Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination. Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Mitsubishi Electric Corporation, Mitsubishi Electric Semiconductor Systems Corporation or an authorized Mitsubishi Semiconductor product distributor for further details on these materials or the products contained therein.

For inquiries about the contents of this manual or software, email or fax using a text file the installer generates in the following directory or "Technical Support Communication Sheet" included in the manual or User's Manual to your nearest Mitsubishi office or its distributor. When sending email, write the same items of the "Technical Support Communication Sheet".

\SUPPORT\Product-name\SUPPORT.TXT

Mitsubishi Tool Homepage http://www.tool-spt.mesc.co.jp/index_e.htm

RENESAS
Renesas Technology Corp.

NC77 V.5.20 User's Manual

Contents

RENESAS
Renesas Technology Corp.

Contents

Preface

NC77 is the C compiler for the Mitsubishi 7700 16-bit microcomputer family (7700, 7750, 7751, 7770, and 7790 Series). NC77 converts programs written in C into assembly language source files for the 7700 family. You can also specify compiler options for assembling and linking to generate hexadecimal files that can be written to the microcomputer.

Terminology

The following terms are used in the NC77 V.5.20 User Manuals.

Term	Meaning
nc77	Compile driver and its executable file
NC77	The NC77 V.5.20 product and a collective term for the package including all software
rasm77	Relocatable macro assembler and its executable file
RASM77	The RASM77 product and a collective term for the package including all software
Assembler function	A subroutine written in assembly language
asm function	Assembly language routine written in a C program using the NC77 extended functions
In-line assembly	

Description of Symbols

The following symbols are used in the NC77 V.5.20 manuals:

Symbol	Description
#	Root user prompt
%	UNIX prompt
A>	MS-Windows(MS-DOS) prompt
<RET>	Return key
< >	Mandatory item
[ ]	Optional item
Δ	Space or tab code (mandatory)
▲	Space or tab code (optional)
: (omitted) :	Indicates that part of file listing has been omitted

Additional descriptions are provided where other symbols are used.

NC77 V.5.20

User's Manual

Renesas Technology Corp.

Chapter 1
Introduction to NC77

This chapter introduces the processing of compiling performed by NC77, and provides an example of program development using NC77.

1.1 NC77 Components

NC77 consists of the following four executable files:

1.nc77 .... Compile driver
2.cpp77 .... Preprocessor
3.ccom77 ....Compiler
4.loop77 .... Branch optimizer
5.s2ie .... IEEE-695 absolute format file converter
6.stk77 .... Stack size calculation utility

1.2 NC77 Processing Flow

Figure 1.1 illustrates the NC77 processing flow.

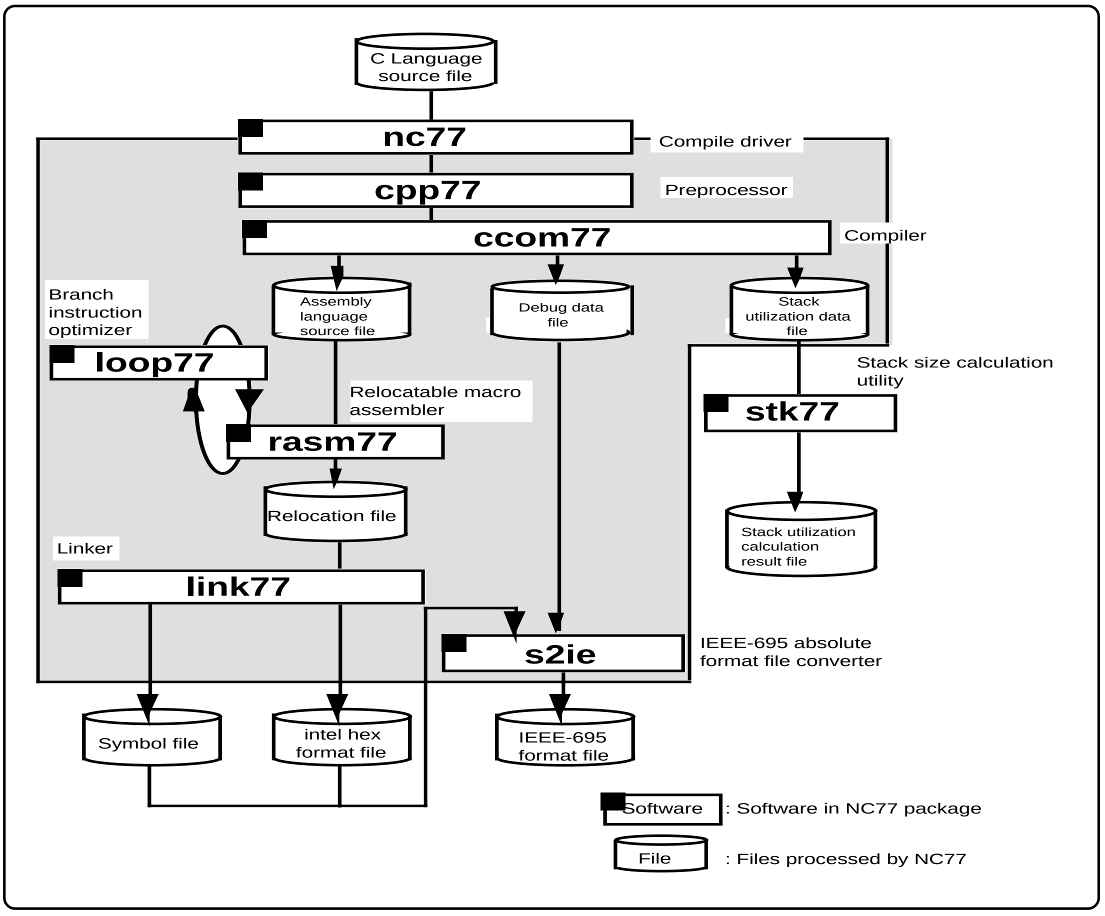


Figure1.1 NC77 Processing Flow

1.2.1 nc77

nc77 is the executable file of the compile driver. By specifying options, nc77 can perform the series of operations from compiling to linking. You can also specify for the rasm77 relocatable macro assembler and four for the link77 linkage editor by including the -rasm77 and -link77 command line options when you start nc77.

1.2.2 cpp77

cpp77 is the executable file for the preprocessor. cpp77 processes macros starting with # (#define, #include, etc.) and performs conditional compiling (#if-#else-#endif, etc.).

1.2.3 ccom77

ccom77 is the executable file of the compiler itself. C source programs processed by cpp77 are converted to assembly language source programs that can be processed by rasm77.

1.2.4 loop77

loop77 is the executable file of the branch optimizer. loop77 optimizes[*1] the branch instructions in assembly language source programs converted by ccom77.

1.2.5 s2ie

2ie is the executable file for the converter that converts the Mitsubishi symbol file format to IEEE-695 absolute format[*2].

1.2.6 stk77

stk77 is the executable file for the stack size calculation utility. stk77 processes the stack utilization display files (extension .stk) generated for each source file by specifying the nc77 command line option -fSSU (-fshow_stack_usage), and outputs data on the relationship between the stack size and C function calls to a calculation result file (extension .siz).

*1. loop77 changes branch instructions (BRA, BCC, etc.) that would result in errors if, for example, the destination jump address is outside the range of relative values, into jump instructions (JMP, etc.) according to the nc77 branch optimization -OB1, -OB2, -OB3 conversion rules, then reassembles the file.

*2. When IEEE-695 absolute format files are read by third-party emulators or simulators, etc., there is a risk that, because of differences such as the existence of data not stipulated by IEEE-695, some functions do not operate correctly or cannot be read. Please note that Mitsubishi Electric Semiconductor Systems Corp. may not be able to resolve such problems. Please see the Release Notes supplied with the NC77 package for details of the operating environment.

1.3 Example Program Development

Figure 1.2 shows the flow for the example program development using NC77. The program is described below. (Items [1] to [4] correspond to the same numbers in Figure 1.2.)

[1]The C source program AA.c is compiled using nc77, then assembled using rasm77 to create the relocatable object file AA.r77.
[2]The startup program ncrt0.a77 and the include file section.inc, which contains information on the sections, are matched to the system by altering the section mapping, section size, and interrupt vector table settings.
[3]The modified startup program is assembled to create the relocatable object file ncrt0.a77.
[4]The two relocatable object files AA.r77 and ncrt0.a77 are linked by the linkage editor link77, which is run from nc77, to create the absolute module file AA.hex.

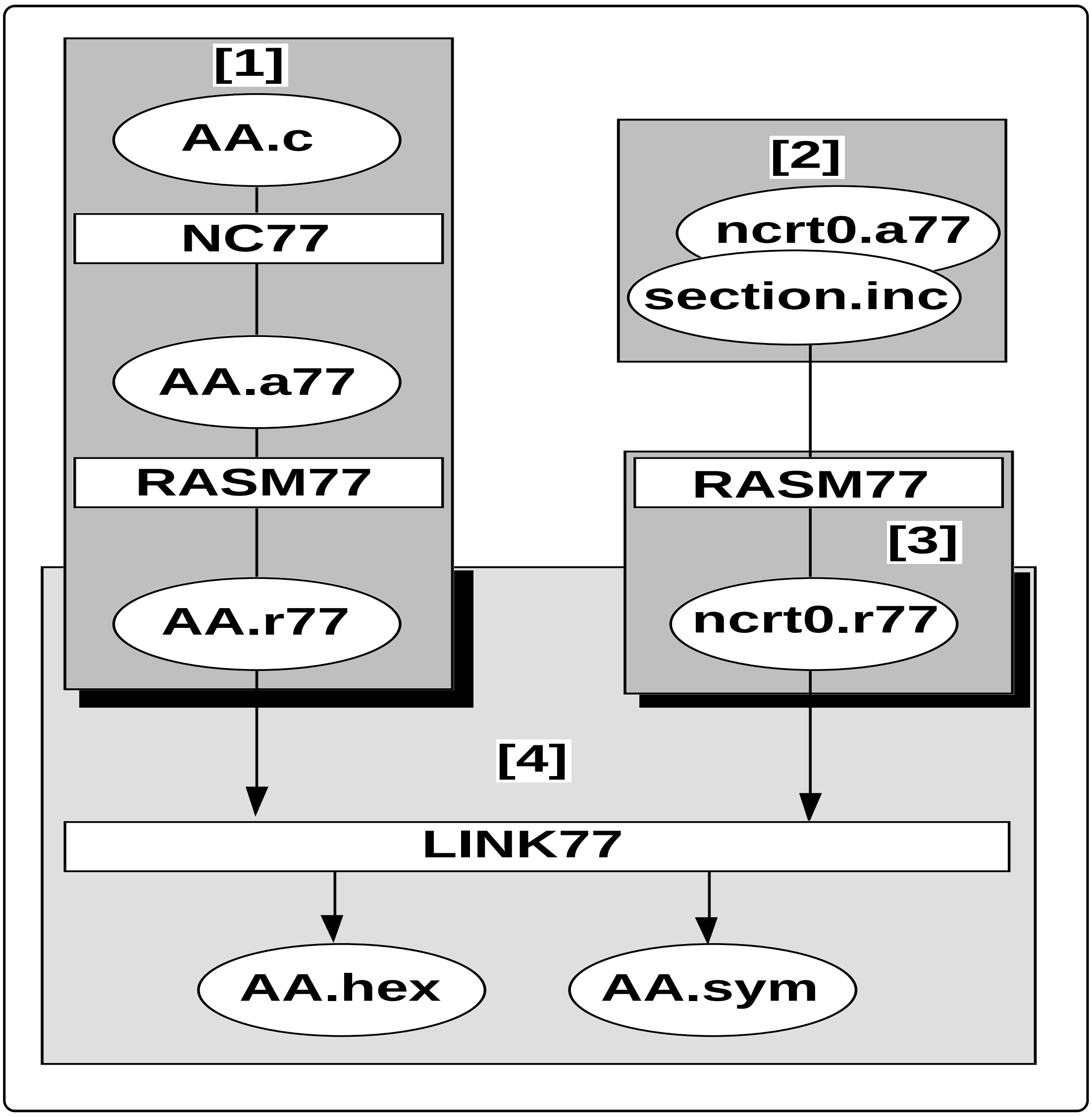


Figure 1.2 Program Development Flow

Figure 1.3 is an example make file containing the series of operations shown in Figure 1.2.

```
AA.hex : ncrt0.a77 AA.r77
        nc77 -oAA ncrt0.r77 AA.r77

ncrt0.r77 : ncrt0.a77
        rasm77 ncrt0.a77

AA.r77 : AA.c
        nc77 -c AA.c
```

Figure 1.3 Example make File

Figure 1.4 shows the command line required for nc77 to perform the same operations as in the makefile shown in Figure 1.3.

```
% nc77 -oAA ncrt0.a77 AA.c<RET>
```

% : Indicates the prompt
<RET> : Indicates the Return key

*Specify ncrt0.a77 first ,when linking.

Figure 1.4 Example nc77 Command Line

1.4 NC77 Output Files

This chapter introduces the preprocess result C source program output when the sample program smp.c is compiled using NC77, the assembly language source program, and the stack utilization display file.

1.4.1 Introduction to Output Files

With the specified command line options, the nc77 compile driver outputs the files shown in Figure 1.5. Below, we show the contents of the files output when the C source file smp.c shown in Figure 1.6 is compiled, assembled, and linked.

See the RASM77 User Manual for the relocatable object files (extension .r77), print files (extension .prn),and map files (extension .map) output by rasm77 and link77.

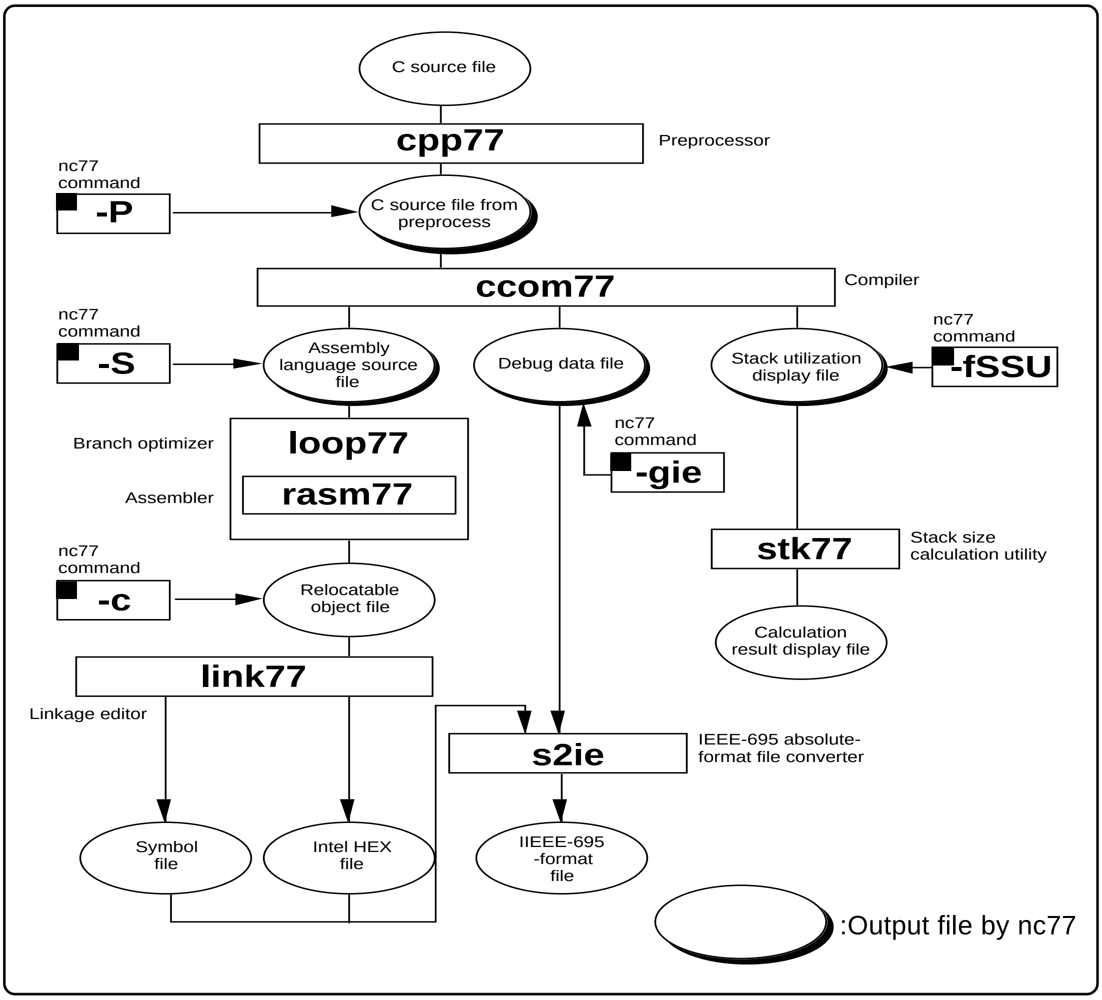


Figure 1.5 Relationship of nc77 Command Line Options and Output Files

```
#include <stdio.h>
#define CLR     0
#define PRN     1

void main()
{
          int flag;
          flag = CLR;

#ifdef PRN
           printf("flag = %d\n",flag);
#endif
}
```

Figure 1.6 Example C Source File (smp.c)

1.4.2 Preprocessed C Source Files

The cpp77 processes preprocess commands starting with #. Such operations include header file contents, macro expansion, and judgements on conditional compiling.

The C source files output by the preprocessor include the results of cpp77 processing of the C source files. Therefore, do not contain preprocess lines other than #pragma and #line. You can refer to these files to check the contents of programs processed by the compiler. The file extension is .i.

Figures 1.7 and 1.8 are examples of file output.

```
                                                                  [1]
typedef struct _iobuf {
    char _buff;
    int _cnt;
    int _flag;
    int _mod;
    int (* _func_in)();
    int (* _func_out)();
} FILE;
          :
      (omitted)
          :
typedef long fpos_t;

typedef unsigned int size_t;

extern FILE _iob[];
```

Figure 1.7 Example Preprocessed C Source File (1) (smp.i)

```
int getc(FILE  *st);                                                    [1]
int getchar(void);
int putc(int c, FILE  *st);
int putchar(int c);
int feof(FILE  *st);
int ferror(FILE  *st);
int fgetc(FILE  *st);
char * fgets(char  *s, int n, FILE  *st);
int fputc(int c, FILE  *st);
int fputs(const char  *s, FILE  *st);
size_t fread(void  *ptr, size_t size, size_t nelem, FILE  *st);
          :
      (omitted)
          :
int ungetc(int c, FILE  *st);
int printf(const char  *format, ...);
int fprintf(FILE  *st, const char  *format, ...);
int sprintf(char  *s, const char  *format, ...);
          :
      (omitted)
          :
extern  int     init_dev(FILE  *, int);
extern  int     speed(int, int, int, int);
extern  int     init_prn(void);
extern  int     _sget(void);
extern  int     _sput(int);
extern  int     _pput(int);
extern  char     *_print( int(*)(), char  *, int **, int * );
```

```
void main()                                  [2]
{
        int flag;
        flag = 0 ;   ⇐[3]


         printf("flag = %d\n",flag);  ⇐[4]

}
```

Figure 1.8 Example Preprocessed C Source File (2) (smp.i)

Let's look at the contents of the preprocessed C source file.
Items [1] to [4] correspond to [1] to [4] in Figures 1.7 and 1.8.

[1]Shows the expansion of header file stdio.h specified in #include
[2]Shows the C source program resulting from expanding the macro
[3]Shows that CLR specified in #define is expanded as 0
[4]Shows that, because PRN specified in #define is 1, the compile condition is satisfied and the printf function is output

1.4.3 Assembly Language Source Files

The assembly language source file is a file that can be processed by RASM77 as a result of the compiler ccom77 converting the preprocess result C source file. The output files are assembly language source files with the extension .a77

Figures 1.9 and 1.10 are examples of the output files. When the nc77 command line option -dsource (-dS) is specified, the assembly language source files contain the contents of the C source file as comments.

```
    .language   c

;## NC77 Compiler for 7700 Family OUTPUT
;## ccom77 2.02.05
;## Copyright(c) 1999 MITSUBISHI ELECTRIC CORPORATION
;## and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;## All Rights Reserved
;## Compile Start Time Mon Feb  1 11:34:21 1999
;## COMMAND_LINE: ccom77  /home/kawajiri/tmp/2177xxx.i -o ./xxx.a77 -dS
;## Normal Optimize   OFF                                                   [1]
;## ROM size Optimize OFF
;## Speed Optimize    OFF
;## Default function is     far
;## Default ROM is    near
;## Default RAM is    near
;## MCU type is             M37700
;## Branch instruction is   "bra"

    .MCU        M37700
    .pointer    2
    .include    ./xxx.ext

__DT      .EQU        0     ; data bank
__STACK   .EQU        0     ; stack bank


;## #     FUNCTION main
;## #     FRAME AUTO  (    flag)  size   2,   offset 1
;## #     ARG Size(0) Auto Size(2)      Context Size(5)

    .source xxx.c
    .section    program_F
;## # C_SRC :   {
    .DT   __DT
    .DP   OFF
    .func _main
    .pub  _main
_main:
    phd
    pha
    tsa
    tad
    .cline      8
;## # C_SRC :         flag = CLR;
    ldm.W #0000H,DP:1 ;  flag
```

Figure 1.9 Example Assembly Language Source File "smp.a77" (1/2)

```
    .cline      11
;## # C_SRC :         printf("flag = %d\n", flag);     ⇐[2]
    pei   #1    ;  flag
    pea   #OFFSET ___T0
    jsrl  _printf
    plx
    plx
    .cline      13
;## # C_SRC :   }
    plx
    pld
    rtl
    .endfunc    _main

    .SECTION    rom_NO
___T0:
    .byte 66H   ;  'f'
    .byte 6cH   ;  'l'
    .byte 61H   ;  'a'
    .byte 67H   ;  'g'
    .byte 20H   ;  ' '
    .byte 3dH   ;  '='
    .byte 20H   ;  ' '
    .byte 25H   ;  '%'
    .byte 64H   ;  'd'
    .byte 0aH
    .byte 00H
    .END

;## Compile End Time Mon Feb  1 11:34:22 1999
```

Figure 1.10 Example Assembly Language Source File "smp.a77" (2/2)

Let's look at the contents of the assembly language source files. Items [1] to [2] correspond to [1] to [2] in Figure 1.9 and Figure 1.10.

[1]Shows status of optimization option, and information on the initial settings of the near and far attribute for ROM and RAM.

[2]When the nc77 command line option -dsource (-dS) is specified, shows the contents of the C source file(s) as comments

Chapter 2
Basic Method for Using the Compiler

This chapter describes how to start the compile driver nc77 and the command line options.

2.1 Starting Up the Compiler

2.1.1 nc77 Command Format

The nc77 compile driver starts the compiler commands (cpp77, ccom77, loop77, and s2ie) ,the assemble command rasm77 and the link command link77 to create a absolute module file. The following information (input parameters) is needed in order to start nc77:

1. C source file(s)
2. Assembly language source file(s)
3. Relocatable object file(s)
4. Command line options (optional)

These items are specified on the command line.

Figure 2.1 shows the command line format. Figure 2.2 is an example. In the example, the following is performed:

1. Startup program nart0.a77 is assembled;
2. C source program sample.c is compiled and assembled;
3. Relocatable object files nart0.a77 and sample.r77 are linked.

The machine language data file sample.hex is also created. The following command line options are used:

*Specifies machine language data file sample.hex .... -o
*Specifies output of list file (extension .lst) at assembling .... -rasm77 "-l"
*Specifies output of map file (extension .map) at linking .... -link77 "-ms"

*1.The rasm77 assemble command is invoked from the loop77 branch optimizer. RASM77 is not directly started from the nc77 compile driver.

```
% nc77Δ[command-line-option]Δ[assembly-language-source-file-name]Δ
        [relocatable-object-file-name]Δ<C-source-file-name>

%       : Prompt
< >     : Mandatory item
[ ]     : Optional item
Δ       : Space
```

Figure 2.1 nc77 Command Line Format

```
% nc77 -osample -rasm77 "-l" -link77 "-ms" nart0.a77 sample.c<RET>

<RET> : Return key
 * Always specify the startup program first when linking.
```

Figure 2.2 Example nc77 Command Line

2.1.2 Command File

When invoking nc77, one or more command options listed in a command file (a text file) can be specified by one parameter.

a. Command file input format

```
% nc77Δ[command-line-option]Δ<@file-name>[command-line-option]Δ

%       : Prompt
< >     : Mandatory item
[ ]     : Optional item
Δ       : Space
```

Figure 2.3 Command File Command Line Format

```
% nc77 -c @test.cmd -g<RET>

<RET> : Return key
 * Always specify the startup program first when linking.
```

Figure 2.4 Example Command File Command Line

Command files are written in the manner described below.

```
                                      test.cmd<CR>
                                      nart0.a77<CR>
Command File description  ------->    sample1.c sample2.r77<CR>
                                      -g -rasm77 -l<CR>
                                      -o<CR>
                                      sample<CR>
 <CR>: Denotes carriage return.
```

Figure 2.5 Example Command File description

b. Rules on command file description

The following rules apply for command file description.

- Only one command file can be specified at a time. You cannot specify multiple command files simultaneously.
- No command file can be specified in another command file.
- Multiple command lines can be written in a command file.
- New-line characters in a command file are replaced with space characters.
- The maximum number of characters that can be written in one line of a command file is 2,048. An error results when this limit is exceeded.

c. Precautions to be observed when using a command file

A directory path can be specified for command file names. An error results if the file does not exist in the specified directory path.

Command files for link77 whose file name extension is ".cm$" are automatically generated in order for specifying files when linking. Therefore, existing files with the file name extension ".cm$," if any, will be overwritten. Do not use files which bear the file name extension ".cm$" along with this compiler. You cannot specify two or more command files simultaneously. If multiple files are specified, the compiler displays an error message "Too many command files."

2.1.3 Notes on NC77 Command Line Options

a. Notes on Coding nc77 Command Line Options

The nc77 command line options differ according to whether they are written in uppercase or lowercase letters. Some options will not work if they are specified in the wrong case.

b. Priority of Options for Controlling nc77

If you specify both the following options in the nc77 command line, the -S option takes precedence and only the assembly language source files will be generated.

- -c : Stop after creating relocatable files (extensions .r77)
- -S : Stop after creating assembly language source files (extensions .a77)

c. Combination of Optimization Options

If you specify both -OS and -OB2, or -OS and -OB3 optimization options, NC77 generates JMP or JMPL instructions but no BRA instruction.

- -OS : Speed takes precedence over ROM size.
- -OB2 : Optimization of branch instructions to within same bank, with speed a priority
- -OB3 : Optimization of branch instructions to outside the bank, with speed a priority

2.1.4 nc77 Command Line Options

a. Options for Controlling Compile Driver

Table 2.1 shows the command line options for controlling the compile driver.

Table 2.1 Options for Controlling Compile Driver

Option	Function
-c	Creates a relocatable file (extension .r77) and ends processing [*1]
-D*identifier*	Defines an identifier. Same function as #define.
-I*directory*	Specifies the directory containing the file(s) specified in #include. You can specify up to 8 directories.
-E	Invokes only preprocess commands and outputs result to standard output.[*1]
-P	Invokes only preprocess commands and creates a file (extension .i). [*1]
-S	Creates an assembly language source file (extension .a77) and ends processing.[*1]
-U *predefined macro*	Undefines the specified predefined macro.
-silent	Suppresses the copyright message display at startup.
-M60 (NC30 Only)	Generates object code for M30600(M16C/60).
-M61 (NC30 Only)	Generates object code for M30610(M16C/61).
-M62E (NC30 Only)	Generates object code for using the extended memory area of M30620(M16C/62).

b. Options Specifying Output Files

Table 2.2 shows the command line option that specifies the name of the output machine language data file.

Table 2.2 Options for Specifying Output Files

Option	Function
-o*filename*	Specifies the name(s) of the file(s) (absolute module file, map file, etc.) generated by link77. This option can also be used to specify the destination directory. Do not specify the filename extension.
-dir	Specifies the destination directory of the file(s) (absolute module file, map file, etc.) generated by link77.

c. Version Information Display Option

Table 2.3 shows the command line options that display the cross-tool version data.

Table 2.3 Options for Displaying Version Data

Option	Function
-v	Displays the name of the command program and the command line during execution
-V	Displays the startup messages of the compiler programs, then finishes processing (without compiling)

1. If you do not specify command line options -c, -E, -P, or -S, nc77 finishes at link77 and output files up to the absolute load module file (extension .hex) are created.

d. Options for Debugging

Table 2.4 shows the command line options for outputting the symbol file for the C source file.

Table 2.4 Options for Debugging

Option	Short form	Function
-g	None.	Outputs the symbol file (extension .sym) required for debugging
-genter	None.	generates a stack flame at calling a function
-gie	None.	Outputs an IEEE-695 absolute format file (extension .ie)
-gie_no_local_symbol	-gINLS	Outputs a file in absolute IEEE-695 format (having the extension .ie), but doesn't output local symbols contained in the assembly language file to the IEEE-695 file

e. Optimization Options

Table 2.5 shows the command line options for optimizing program execution speed and ROM capacity.

Table 2.5 Optimization Options

Option	Short form	Function
-O[1-5]	None.	Maximum optimization of speed and ROM size
-OR	None.	Maximum optimization of ROM size followed by speed
-OS	None.	Maximum optimization of speed followed by ROM size
-Oconst	-OC	Performs optimization by replacing references to the const-qualified external variables with constants
-Ono_bit	-ONB	Suppresses optimization based on grouping of bit manipulations
-Ono_break_source_debug	-ONBSD	Suppresses optimization that affects source line data
-Ono_float_const_fold	-ONFCF	Suppresses the constant folding processing of floating point numbers
-Ono_stdlib	-ONS	Inhibits inline padding of standard library functions and modification of library functions.
-Osp_adjust	-OSA	Optimizes removal of stack correction code. This allows the necessary ROM capacity to be reduced. However, this may result in an increased amount of stack being used.
-Ostack_frame_align	-OSFA	Aligns the stack frame on an even boundary.

f. Generated Code Modification Options

Table 2.6 shows the command line options for controlling nc77-generated assembly code.

Table 2.6 (1/2) Generated Code Modification Options

Option	Short form	Description
-fansi	None.	Makes -fnot_reserve_far_and_near, -fnot_reserve_asm, -fnot_reserve_inline, and -fextend_to_int valid.
-fnot_reserve_asm	-fNRA	Exclude asm from reserved words. (Only _asm is valid.)
-fnot_reserve_far_and_near	-fNRFAN	Exclude far and near from reserved words. (Only _far and _near are valid.)
-fnot_reserve_inline	-fNRI	Exclude far and near from reserved words. (Only _inline is made a reserved word.)
-fextend_to_int	-fETI	Performs operation after extending char-type data to the int type. (Extended according to ANSI standards.)*1
-fchar_enumerator	-fCE	Handles the enumerator type as an unsigned char type, not as an int type.
-fno_even	-fNE	Allocate all data to the odd section , with no separating odd data from even data when outputting .
-fshow_stack_usage	-fSSU	Outputs the usage condition of the stack pointer to a file (extension .stk).
-ffar_RAM_data	-fFRAM	Changes the default attribute of RAM data to far.
-ffar_ROM_data	-fFROM	Changes the default attribute of ROM data to far.
-fall_far	-fAF	Changes all defaults to far types.
-fnear_function	-fNF	Sets the function default to near. Near functions are called with jsr and returned with rts.
-fnot_use_MVN	-fNUM	Suppresses transfer of blocks with the MVN instruction (The MVN instruction is used for assignment among structures.)
-bank=*bank No.*	None.	Specifies the value of the data bank register (DT) at compiling. The default when not specified is 0.
-fswitch_table	-fST	Uses the jump table only when the code size of case statements in switch statements is satisfactory.
-fconst_not_ROM	-fCNR	Does not handle the types specified by const as ROM data.
-fnot_address_volatile	-fNAV	Does not regard the variables specified by #pragma ADDRESS (#pragma EQU) as those specified by volatile.
-fsmall_array	-fSA	When referencing a far-type array, this option calculates subscripts in 16 bits if the total size of the array is within 64K bytes.

*1. char-type data or signed char-type data evaluated under ANSI rules is always extended to int-type data. This is because operations on char types (c1=c2*2/c3; for example) would otherwise result in an overflow and failure to obtain the intended result.

Table 2.6 (2/2) Generated Code Modification Options

Option	Short form	Description
-fenable_register	-fER	Make register storage class available.
-fuse_DIV	-fUD	This option changes generated code for divide operation.

g. Warning Options

Table 2.6 shows the command line options for outputting warning messages for contraventions of nc77 language specifications.

Table 2.7 Warning Options

Option	Short form	Function
-Wnon_prototype	-WNP	Outputs warning messages for functions without prototype declarations.
-Wunknown_pragma	-WUP	Outputs warning messages for non-supported #pragma.
-Wno_stop	-WNS	Prevents the compiler stopping when an error occurs.
-Wstdout	None.	Outputs error messages to the host machine's standard output (stdout).
-Werror_file<file name>	-WEF	Outputs error messages to the specified file.
-Wstop_at_warning	-WSAW	Stops the compiling process when a warning occurs.
-Wnesting_comment	-WNC	Outputs a warning for a comment including */ .
-Wccom_max_warnings	-WCMW	This option allows you to specify an upper limit for the number of warnings output by ccom77.
-Wall	None.	Displays message for all detectable warnings.
-Wmake_tagfile	-WMT	Outputs error messages to the tag file of source-file by source-file.

h. Assemble and Link Options

Table 2.8 shows the command line options for specifying RASM77 and LINK77 options.

Table 2.8 Assemble and Link Options

Option	Function
-rasm77Δ<"*option(s)*">	Specifies options for the rasm77 link command. You can specify a maximum of 4 options. If you specify two or more options, enclose them in double quotes.
-link77Δ<"*option(s)*">	Specifies options for the link77 assemble command. You can specify a maximum of 4 options. If you specify two or more options, enclose them in double quotes.

i. 7750/7751-Compatible Code Generation Option

Table 2.9 shows the command line option for specifying that NC77 generates 7750/7751-compatible code.

Table 2.9 7750/7751-Compatible Code Generation Option

Option	Shortform	Function
-m7750	None.	Generates code that is compatible with the 7750/7751 series

j. Miscellaneous Option

Table 2.10 shows the command line option for processing the assembly language source files generated by nc77.

Table 2.10 Miscellaneous Option

Option	Short form	Function
-dsource	-dS	Outputs the C source listings in the output assembly language source list as comments

2.2 Preparing the 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 (7700), 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.

The following explains about the startup program and describes how to customize it.

2.2.1 Sample of Startup Program

The NC77 startup program consists of the following two files:

1. ncrt0.a77

This program is run at the start of the program or immediately after a reset.

2. section.inc

This program is included from ncrt0.a77.

Figures 2.6 to 2.9 show the ncrt0.a77 source program list. Figures 2.10 to 2.13 show the section.inc source program list.

```
;***************************************************************************
;
;   NC77 COMPILER for 7700 FAMILY
;   Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
;   AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;   All Rights Reserved.
;
;   ncrt0.a77 : NC77 startup program
;
;   This program is applicable when using the basic I/O library
;
;
;
;***************************************************************************
    .include section.inc                              ⇐[1]
;-------------------------------------------------------------------
; section-size
;-------------------------------------------------------------------
    .section    rom_NE
    .pub  bss_NEsz
    .pub  bss_NOsz
    .pub  bss_FEsz
    .pub  bss_FOsz
    .pub  data_NEsz
    .pub  data_NOsz
    .pub  data_FEsz
    .pub  data_FOsz
bss_NEsz: .dword      sizeof bss_NE
bss_NOsz: .dword      sizeof bss_NO
bss_FEsz: .dword sizeof bss_FE
bss_FOsz: .dword sizeof bss_FO
data_NEsz:     .dword      sizeof data_NE
data_NOsz:     .dword      sizeof data_NO
data_FOsz:     .dword      sizeof data_FO
data_FEsz:     .dword      sizeof data_FE
```

[1] Includes section.inc

Figure 2.6 Startup Program List (1)(ncrt0.a77 1/4)

```
;=====================================================================
; Initialize Macro declaration
;----------------------------------------------------------------------
BZERO:    .macro     SIZE,TOP
    lda   A,DT:SIZE + 2
    pha
    lda   A,DT:SIZE
    pha
    pea   #bank TOP
    pea   #offset TOP
    .ext  _bzero
    jsrl  _bzero
    .endm

BCOPY:    .macro     SIZE,TO,FROM
    lda   A,DT:SIZE + 2
    pha
    lda   A,DT:SIZE
    pha
    pea   #bank TO
    pea   #offset TO
    pea   #bank FROM
    pea   #offset FROM
    .ext  _bcopy
    jsrl  _bcopy
    .endm

;=====================================================================
; Libraly file name definition
;----------------------------------------------------------------------
    .lib  nc77lib

;=====================================================================
; Interrupt section start
;----------------------------------------------------------------------
    .pub  start
    .section   interrupt
start:                                                  ⇐[2]
;----------------------------------------------------------------------
; after reset,this program will start
;----------------------------------------------------------------------
__DT      .equ       00h                                ⇐[3]
    ldt   #__DT      ; Initialize data bank register
    sem
    ldm.B #24H,DT:5eH      ; set processor mode register              ⇐[4]
    clp   m,x,d
    lda.w a, #offset stack_top-1 ;    Initialize stack pointer        ⇐[5]
    tas

;=====================================================================
; NEAR area initialize.
;---------------------------------------------------------------------
; bss_NE & bss_NO zero clear
;---------------------------------------------------------------------
    BZERO bss_NEsz,bss_NE_top                           ⇐[6]
    BZERO bss_NOsz,bss_NO_top
;----------------------------------------------------------------------
; Copy data_NE(NO) section from data_INE(INO) section
;----------------------------------------------------------------------
    BCOPY data_NEsz,data_NE_top,data_INE_top            ⇐[7]
    BCOPY data_NOsz,data_NO_top,data_INO_top
    lda.w a,#offset stack_top - 1
    tas
```

[2] After a reset, execution starts from this label (start)

[3] Changes the _DT value when using the NC77 command line option -bank=n

[4] Sets processor operating mode

[5] Initializes the stack pointer

[6] Clears the bss section on the near area (to zeros)

[7] Moves the initializer for the data section on the near area to the RAM area

Figure 2.7 Startup Program List (2) (ncrt0.a77 2/4)

```
;=====================================================================
; FAR area initialize.
;---------------------------------------------------------------------
; bss_FE & bss_FO zero clear
;---------------------------------------------------------------------
    BZERO bss_FEsz,bss_FE_top                          ⇐[8]
    BZERO bss_FOsz,bss_FO_top
;---------------------------------------------------------------------
; Copy data_FE(FO) section from data_IFE(IFO) section
;---------------------------------------------------------------------
    BCOPY data_FEsz,data_FE_top,data_IFE_top           ⇐[9]
    BCOPY data_FOsz,data_FO_top,data_IFO_top
    lda.w a,#offset stack_top - 1
    tas

;=====================================================================
; heap initialize
;---------------------------------------------------------------------
    .ext  __mbase, __mnext, __msize                    ⇐[10]
    lda.w a, #offset heap_top
    lda.w b, #bank heap_top
    sta   a, __mbase
    sta   , __mbase + 0002h
    sta   a, __mnext
    sta   b, __mnext + 0002h
    lda.w a, #offset HEAPSIZE
    lda.w b, #bank HEAPSIZE
    sta   a, __msize
    sta   b, __msize + 0002h

;
;
;=====================================================================
; Initialize standard I/O
;---------------------------------------------------------------------
    .ext  _init
    jsrl  _init                                        ⇐[11]

;=====================================================================
; Call main() function
;---------------------------------------------------------------------
    .ext  _main
    jsrl  _main                                        ⇐[12]
```

[8] Clears the far bss section (to zeros)

[9] Moves the initial values of the far data section to RAM

[10] Initializes the heap area. Comment out this line if no memory management function is used.

[11] Calls the init function, which initializes standard I/O. Comment out this line if no I/O function is used.

[12] Calls the 'main' function. Both M and X flags must be cleared for the main function to be called.

Figure 2.8 Startup Program List (3) (ncrt0.a77 3/4)

```
;====================================================================
; exit() function
;---------------------------------------------------------------------
    .func       _exit                              ⇐[13]
    .pub        _exit
_exit:                   ; End program
    bra         _exit
    .endfunc    _exit
    .func       ?exit
    .pub        ?exit
?exit:                   ; End program
    bra         ?exit
    .endfunc    ?exit

;====================================================================
; dummy interrupt function
;---------------------------------------------------------------------
dummy_int:                                         ⇐[14]
    rti
    .end
;*****************************************************************************
;
;   NC77 COMPILER for 7700 FAMILY
;   Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
;   AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;   All Rights Reserved.
;
;*****************************************************************************

[13] exit function
[14] Dummy interrupt processing function
```

Figure 2.9 Startup Program List (4) (ncrt0.a77 4/4)

```
;***************************************************************************
;
;       NC77 COMPILER for 7700 FAMILY
;       Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
;       AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;       All Rights Reserved.
;
;       section.inc     : section definition
;
;       This program is applicable when using the basic I/O library
;
;
;
;***************************************************************************
;---------------------------------------------------------------------
; Arrangement of section
;---------------------------------------------------------------------
    .section   data_NE
    .org  80H                                      ⇐[1]
data_NE_top:
;
    .section   data_NO
data_NO_top:
;
    .section   bss_NE
bss_NE_top:
;
    .section   bss_NO
bss_NO_top:
;
    .section   stack
    .blkb 300H             ; stack size          ⇐[2]
stack_top:
;
HEAPSIZE  .equ    300h                             ⇐[3]
    .section   heap
heap_top:
    .blkb HEAPSIZE
;
    .section   interrupt
    .ORG  8000H
;
    .section   program_N
;
    .section   rom_NE
rom_NE_top:
;
    .section   rom_NO
rom_NO_top:
;
    .section   data_INE
data_INE_top:
;
    .section   data_INO
data_INO_top:
;
```

[1] Sets the section starting address using pseudo instruction .ORG

[2] Defines the stack size to be used

[3] Defines heap size to be used

Figure 2.10 Startup Program List (5) (section.inc 1/3)

```
    .section   vector              ; Interrupt vector table
    .org  0ffd6H
ADCOMP:                                       ⇐[4]
    .word dummy_int
TRN1:
    .word dummy_int
REC1:
    .word dummy_int
TRN0:
    .word dummy_int
REC0:
    .word dummy_int
BS2I:
    .word dummy_int
BS1I:
    .word dummy_int
BS0I:
    .word dummy_int
TA4I:
    .word dummy_int
TA3I:
    .word dummy_int
TA2I:
    .word dummy_int
TA1I:
    .word dummy_int
TA0I:
    .word dummy_int
INT2:
    .word dummy_int
INT1:
    .word dummy_int
INT0:
    .word dummy_int
WDT:
    .word dummy_int
RESERVED:
    .word dummy_int
BRK:
    .word dummy_int
DIV0:
    .word dummy_int
;
RESET:
    .word offset start
;
;
```

[4] Example interrupt vector address table

Figure 2.11 Startup Program List (6) (section.inc 2/3)

```
    .section    data_FE
    .org  12000H
data_FE_top:
;
    .section    data_FO
data_FO_top:
;
    .section    bss_FE
bss_FE_top:
;
    .section    bss_FO
bss_FO_top:
;
    .section    program_F
    .ORG  18000H
;
    .section    rom_FE
rom_FE_top:
;
    .section    rom_FO
rom_FO_top:
;
    .section    data_IFE
data_IFE_top:
;
    .section    data_IFO
data_IFO_top:

;*****************************************************************************
;
;       NC77 COMPILER for 7700 FAMILY
;       Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
;       AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;       All Rights Reserved.
;
;*****************************************************************************
```

Figure 2.12 Startup Program List (7) (section.inc 3/3)

2.2.2 Customizing the Startup Program

a. Overview of Startup Program Processing

About ncrt0.a77

This program is run at the start of the program or immediately after a reset. It performs the following process mainly:

- ●Sets the bank value (_DT) of the data near area
- ●Sets the processor's operating mode
- ●Initializes the stack pointer
- ●Initializes the data near area

  bss_NE and bss_NO sections are cleared (to 0). Also, the initial values in the ROM area (data_INE, data_INO) are transferred to RAM (data_NE and data_NO).*1
- ●Initializes the data far area

  bss_FE and bss_FO sections are cleared (to 0). Also, the initial values in the ROM area (data_IFE, data_IFO) storing them are transferred to RAM (data_FE and data_FO).*1
- ●Initializes the standard I/O function library
- ●Initializes the heap area
- ●Calls the 'main' function

* 1.For global variables with initial values, NC77 outputs initial values to the RAM area accessed as a variable (data_NE and data_NO) and the ROM area (data_INE and data_INO) storing those initial values. The startup program transfers the initial values to the RAM area.

b. Modifying the Startup Program

Figure 2.14 summarizes the steps required to modify the startup programs to match the target system.

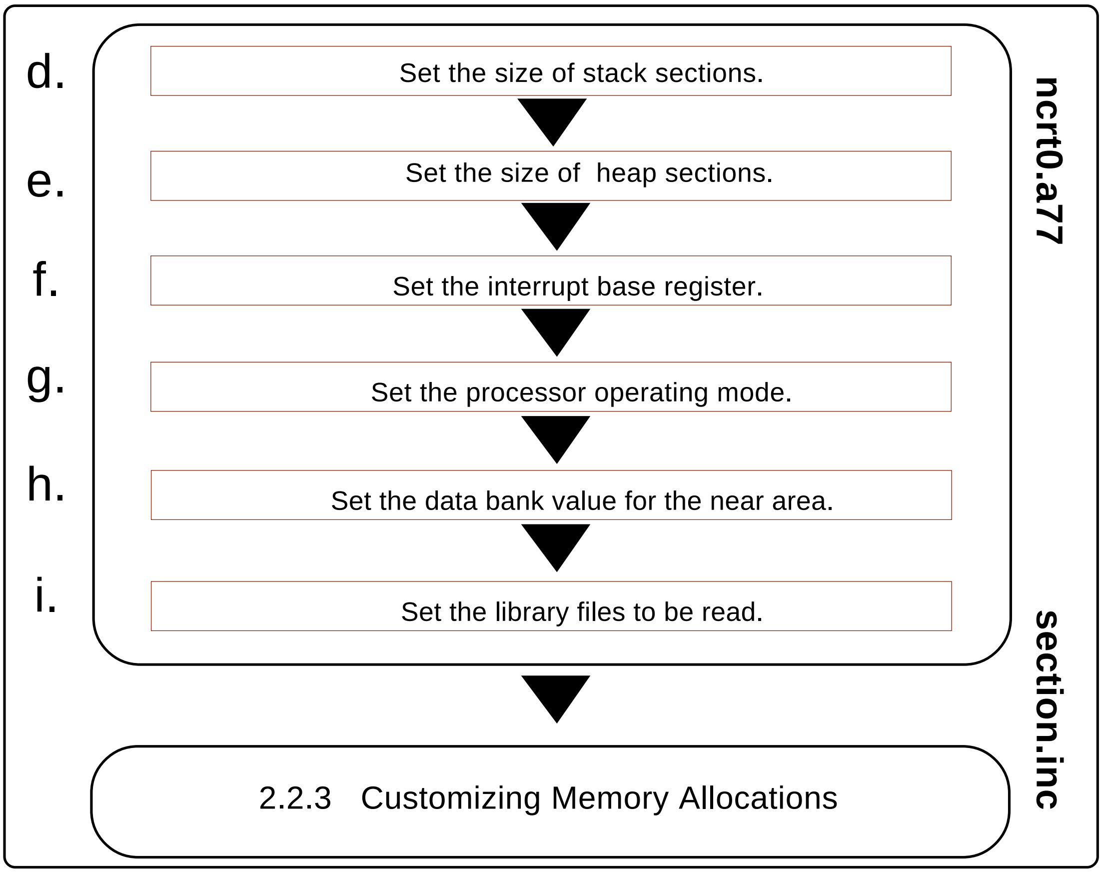


Figure 2.14 Example Sequence for Modifying Startup Programs

c. Examples of startup modifications that require caution

(1) Settings When Not Using Standard I/O Functions

The init function initializes the 7700 Series I/O. It is called before main in ncrt0.a77. Figure 2.15 shows the part where the init function is called.

If your application program does not use standard I/O, comment out the init function call from ncrt0.a77.

```
;=====================================================================
; Initialize standard I/O
;----------------------------------------------------------------------
        .ext     _init
        jsrl    _init

;=====================================================================
; Call main() function
```

Figure 2.15 Part of ncrt0.a77 Where init Function is Called

If you are using only sprintf and sscanf, the init function does not need to be called.

(2) Settings When Not Using Memory Management Functions

To use the memory management functions calloc and malloc, etc., not only is an area allocated in the heap section but the following settings are also made in ncrt0.a77.

(1)Initialization of external variable char *_mbase
(2)Initialization of external variable char *_mnext
  Initializes the heap_top label, which is the starting address of the heap section
(3)Initialization of external variable unsigned_msize
  Initializes the "HEAPSIZE" expression, which sets at "2.2.2 e heap section size".

Figure 2.16 shows the initialization performed in ncrt0.a77.

```
;====================================================================
; heap initialize
;--------------------------------------------------------------------
    .ext  __mbase, __mnext, __msize
    lda.w a, #offset heap_top
    lda.w b, #bank heap_top
    sta   a, __mbase
    sta   , __mbase + 0002h
    sta   a, __mnext
    sta   b, __mnext + 0002h
    lda.w a, #offset HEAPSIZE
    lda.w b, #bank HEAPSIZE
    sta   a, __msize
    sta   b, __msize + 0002h
```

Figure 2.16 Initialization When Using Memory Management Functions (ncrt0.a77)

If you are not using the memory management functions, comment out the whole initialization section. This saves the ROM size by stopping unwanted library items from being linked.

(3) Notes on Writing Initialization Programs

Note the following when writing your own initialization programs to be added to the startup program.

(1) If your initialization program changes the m, x, or D flags, return these flags to the original state where you exit the initialization program. Do not change the contents of the data bank register (DT).

(2) If your initialization program calls a subroutine written in C, note the following two points:
  [1] Call the C subroutine only after clearing the m, x, and D flags.
  [2] Select the JSR and JSRL instructions according to the near or far attribute of the called subroutine.

d. Setting the Stack Section Size

Using NC77, the stack section is used for the following according to function.

- ●Storage of auto variables
- ●Work area for complex operations, etc.
- ●Storage of return addresses for function calls and old frame pointer addresses (DPR)
- ●Storage of function parameters
- ●Storage of internal registers storing 64-bit floating points

Set this section to the maximum stack size used by the program.

The following shows how to determine and set the user stack and the interrupt stack sizes.

Stack size is calculated to use the stack size calculation utility stk77.

For more information, refer to the Appendix "G" the stack size calculation utility stk77.

e. Heap Section Size

Set the heap to the maximum amount of memory allocated using the memory management functions calloc and malloc in the program. Set the heap to 0 if you do not use these memory management functions. Make sure that the heap section does not exceed the physical RAM area.

```
;----------------------------------------------------------------------
; HEAP SIZE definition
;----------------------------------------------------------------------
HEAPSIZE        .equ    300h
```

Figure 2.17 Example of Setting Heap Section Size (ncrt0.a77)

f. Setting the interrupt vector table

Set the top address of the interrupt vector table to the part of Figure 2.18 in ncrt0.a77.

```
.section  vector                  ; Interrupt vector table
.org      0ffd6H
```

Figure 2.18 Example of Setting Top Address of Interrupt Vector Table (ncrt0.a77)

g. Setting the Processor Mode Register

Set the processor operating mode to match the target system at address 5EH (Processor mode register)[*1] in the part of ncrt0.a77 shown in Figure 2.18

```
ldm.B #24H,DT:5eH      ; set processor mode register
```

Figure 2.18 Example Setting of Processor Mode Register (ncrt0.a77)

h. Setting the Data Bank Register

Set the value of the data bank register for the near data area in _DT in the startup program ncrt0.a77 (Figure 2.19). Also, when using the nc77 command line option -bank=n (where n=0 to 255), which sets the near area to other than bank 0, set _DT in ncrt0.a77 to the same value as set in the command line option.

```
start:
;---------------------------------------------------------------------
; after reset,this program will start
;---------------------------------------------------------------------
__DT     .equ    00h
    ldt    #__DT                    ; Initialize data bank register
```

Figure 2.19 Example Setting of Data Bank Register (1) (ncrt0.a77)

Figure 2.20 is an example of how to set the near data area to bank 2. In this case, when all linked C programs are compiled, you must set the nc77 command line option to -bank=2. Note that if you set the near data area to other than bank 0, you will not be able to use some standard functions.

```
start:
;---------------------------------------------------------------------
; after reset,this program will start
;---------------------------------------------------------------------
__DT     .equ            02h
        ldt      #__DT                   ; Initialize data bank register
```

Figure 2.20 Example Setting of Data Bank Register (2) (ncrt0.a77)

* 1.This example setting is written for the M37702 group. See the manual or data book for your machine for the address of the processor mode register and the bit settings.

i. Specifying the Library File

Include the code for reading the NC77 library file in ncrt0.a77 (Figure 2.21). If you are using a different library file, created using the LIB77 librarian, specify it in the startup program using the RASM77 pseudo instruction .LIB. The extension of library files that can be specified in this pseudo instruction is .lib. Specify the library file directory in the LIB77 environment variable.

```
;====================================================================
; Library file name definition
;--------------------------------------------------------------------
    .lib        nc77lib
    .lib        usrlib            ←Specifies user's library usrlib.lib
```

Figure 2.21 Example Specification of User's Library File (ncrt0.a77)

2.2.3 Customizing for NC77 Memory Mapping

a. Structure of Sections

In the case of a native environment compiler, the executable files generated by the compiler are mapped to memory by the operating system, such as UNIX. However, with cross-environment compilers such as NC77, the user must determine the memory mapping.

With NC77, storage class variables, variables with initial values, variables without initial values, character string data, interrupt processing programs, and interrupt vector address tables, etc., are mapped to 7700 series memory as independent sections according to their function. The names of sections consist of a base name and attribute as shown below :

Figure 2.22 Section Names

Section Base Name _ Attribute

Table 2.11 shows Section Base Name and Table 2.12 shows Attributes.

Table 2.11 Section Base Names

Section base name	Content
data	Stores data with initial values
bss	Stores data without initial values
rom	Stores character strings, and data specified in #pragma ROM or with the const modifier
program	Stores programs

Table 2.12 Section Naming Rules

Attribute	Meaning	Target section base name
I	Section containing initial values of data	data
N/F	N...near attribute [*1]	data, bss, rom, program
	F...far attribute [*1]	
E/O	E...Even data size	data, bss, rom
	O...Odd data size	

* 1.near and far are NC77 modifiers, used to clarify the addressing mode.
near.........absolute addressing mode (access up to 64KB)
far............absolute long addressing mode (access over 64KB)

Table 2.13 shows the contents of sections other than those based on the naming rules described above.

Table 2.13   Section Names

Section name	Contents
stack	This area is used as a stack. Allocate this area at bank 0 in the 7700 family.
heap	This memory area is dynamically allocated during program execution by memory management functions (e.g., malloc). This section can be allocated at any desired location of the 7700 memory area.
vector	Contains the contents of the 7700 family interrupt vector table. The address to which the interrupt vector table is mapped varies according to the machine. See the User's manual for your machine for details.
interrupt	Contains the interrupt programs (functions specified in #pragma INTERRUPT, #pragma INTF, and #pragma HANDLER). Map this section to bank 0 in the 7700 family.

These sections are mapped to memory according to the settings in the startup program include file section.inc. You can modify the include file to change the mapping.

Figure 2.23 shows the how the sections are mapped according to the sample startup program's include file section.inc.

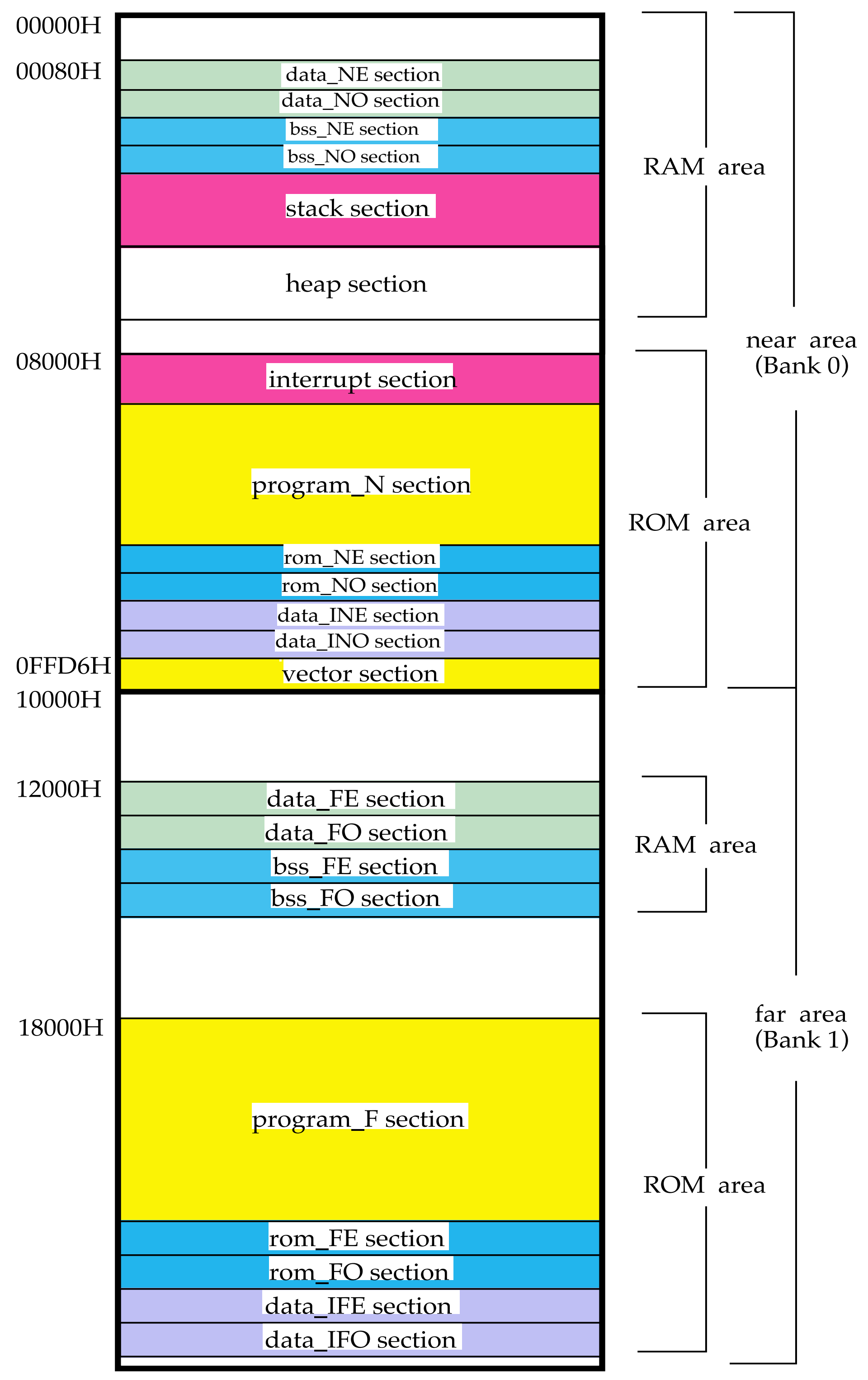


Figure 2.23 Example Section Mapping

b. Outline of memory mapping setup file

About section.inc

This program is included from ncrt0.a77. It performs the following process mainly:

- Maps each section (in sequence)
- Sets the starting addresses of the sections
- Defines the size of the stack and heap sections
- Sets the interrupt vector

c. Modifying the section.inc

Figure 2.24 summarizes the steps required to modify the startup programs to match the target system.

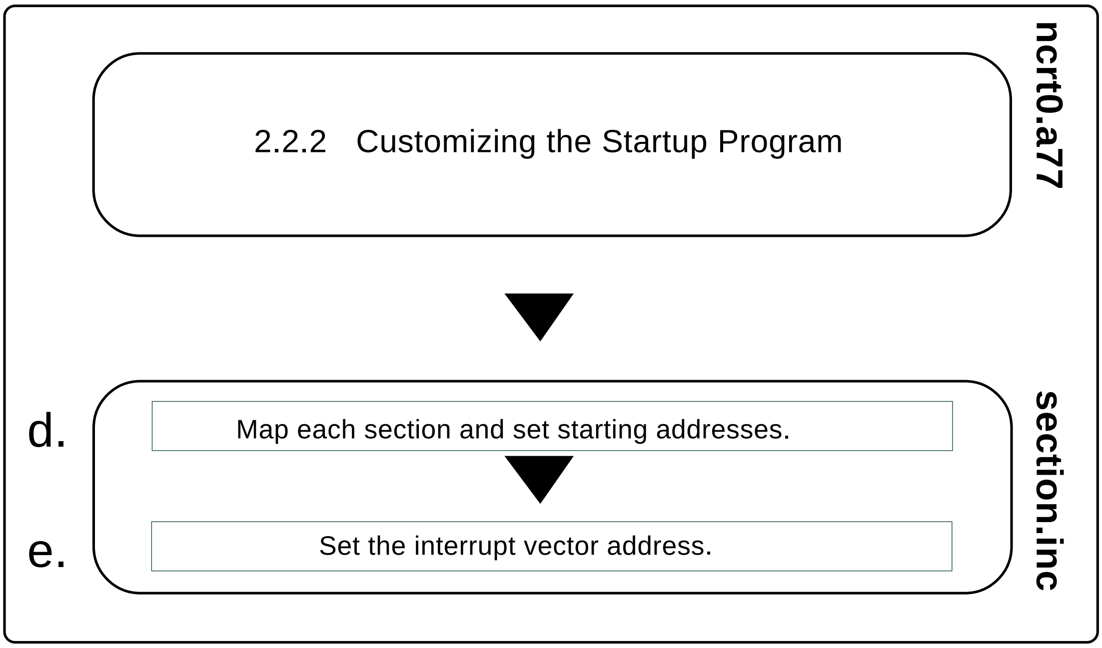


Figure 2.24 Example Sequence for Modifying Startup Programs

d. Mapping Sections and Specifying Starting Address

Map the sections to memory and specify their starting addresses (mapping programs and data to ROM and RAM) in the section.inc include file of the startup program.

The sections are mapped to memory in the order they are defined in section.inc. Use the rasm77 pseudo instruction .ORG to specify their starting addresses. Figure 2.25 is an example of these settings.

```
    .section    program
    .ORG  0C000H        ⇐Specifies the starting address of the program section
;
```

Figure 2.25 Example Setting of Section Starting Address (section. inc)

If no starting address is specified for a section, that section is mapped immediately after the previously defined section.

(1) Rules for Mapping Sections to Memory

Because of the effect on the memory (RAM and ROM) attributes of 7700 series memory, some sections can only be mapped to specific areas. Apply the following rules when mapping sections to memory.

(a)Sections mapped to RAM

- data_NE section
- data_NO section
- data_FE section
- data_FO section
- bss_NE section
- bss_NO section
- bss_FE section
- bss_FO section

- stack section
- heap section

(b)Sections mapped to ROM

- rom_NE section
- rom_NO section
- rom_FE section
- rom_FO section
- data_INE section
- data_INO section
- data_IFE section
- data_IFO section

- program_N section
- interrupt section
- program_Fsection

Note that some sections can only be mapped to specific memory areas in the 7700 family memory space.

(1)Sections mapped only to bank 0

- ●stack section
- ●interrupt section
- ●vector section

(2)Sections mapped to -bank=bank specified in option

●data_NE section	●bss_NO section
●data_NO section	●rom_NE section
●bss_NE section	●rom_NO section

* If you do not specify -bank=, the section is mapped to bank 0.

(3)Sections that can be mapped anywhere in 7700 family address space

●heap section	●program_N section*
●data_INE section	●program_F section
●data_INO section	●rom_FE section
●data_FE section	●rom_FO section
●data_FO section	●data_IFE section
●bss_FE section	●data_IFO section
●bss_FO section	

* The program_N section cannot be mapped across bank boundaries.

If any of the following data sections have a size of 0, they need not be defined. (The section's size can be determined by creating a map file (extension .map) when linking.)

●data_NE, data_INE section	●bss_FE section
●data_NO, data_INO section	●bss_FO section
●data_FE, data_IFE section	●rom_NE section
●data_FO, data_IFO section	●rom_NO section
●bss_NE section	●rom_FE section
●bss_NO section	●rom_FO section

The program_F section contains the runtime library and must therefore be mapped to memory.

(2) Example Section Mapping in Single-Chip Mode

Figures 2.26 and 2.27 are examples of the section.inc include file which is used for mapping sections to memory in single-chip mode. The program for mapping the sections to memory satisfies the following three conditions in this example.

1. Neither data nor programs are mapped outside bank 0.
2. The memory management function library is not used.
3. The -fNF (-fnear_function) option maps all functions to the near area.

```
;;******************************************************************************
;
;       NC77 COMPILER for 7700 Family V.5.00 Release 1
;       Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
;       AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;       All Rights Reserved.
;
;       section.inc     : section definition
;
;       This program is applicable when using the basic I/O library
;
;*****************************************************************************
;---------------------------------------------------------------------
; Arrangement of section
;---------------------------------------------------------------------
        .section        data_NE
        .org    80H
data_NE_top:
;
        .section        data_NO
data_NO_top:
;
        .section        bss_NE
bss_NE_top:
;
        .section        bss_NO
bss_NO_top:
;
        .section        stack
        .blkb           300H                    ; stack size
stack_top:
;
HEAPSIZE        .equ    0h
        .section        heap
heap_top:
        .blkb           HEAPSIZE
;
        .section        interrupt
        .ORG            8000H
;
        .section        program_N
;
        .section        program_F
;
        .section        rom_NE
rom_NE_top:
;
```

←The heap section size is set to 0 because the memory management function is not used.

← Because the runtime library is output to the program_F section, it is necessary to define program_F in single-chip mode.

Figure 2.26 Listing of section.inc in Single-Chip Mode (1/2)

```
        .section        rom_NO
rom_NO_top:
;
        .section        data_INE
data_INE_top:
;
        .section        data_INO
data_INO_top:
;
        .section        vector                  ; Interrupt vector table
        .org    0ffd6H
ADCOMP:
        .word   dummy_int
                :
           (omitted)
                ;
RESET:
        .word   offset start
;
;
        .section        data_FE
        .org    12000H
data_FE_top:
;
        .section        data_FO
data_FO_top:
;
        .section        bss_FE
bss_FE_top:
;
        .section        bss_FO
bss_FO_top:
;
        .section        program_F
        .ORG            18000H
;
        .section        rom_FE
rom_FE_top:
;
        .section        rom_FO
rom_FO_top:
;
        .section        data_IFE
data_IFE_top:
;
        .section        data_IFO
data_IFO_top:

;******************************************************************************
;
;       NC77 COMPILER for 7700 Family V.5.00
;       Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
;       AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
;       All Rights Reserved.
;
;******************************************************************************
```

←You can remove this part, if the section size equal zero.

You also need to remove the initialize program in the far area of ncrt0.a77.

Figure 2.27 Listing of section.inc in Single-Chip Mode (2/2)

e. Setting Interrupt Vector Address

If your program uses interrupt processing, change the interrupt vector address table in the vector section of section.inc. Figure 2.28 is an example of an interrupt vector address table.

```
 .section        vector                         ; Interrupt vector table
        .org    0ffd6H
ADCOMP:                                        ←ADC interrupt
        .word   dummy_int
TRN1:                                          ←UART1 send interrupt
        .word   dummy_int
REC1:                                          ←UART1 receive interrupt
        .word   dummy_int
TRN0:                                          ←UART0 send interrupt
        .word   dummy_int
REC0:                                          ←UART0 receive interrupt
        .word   dummy_int
BS2I:                                          ←Timer B2 interrupt
        .word   dummy_int
BS1I:                                          ←Timer B1 interrupt
        .word   dummy_int
BS0I:                                          ←Timer B0 interrupt
        .word   dummy_int
TA4I:                                          ←Timer A4 interrupt
        .word   dummy_int
TA3I:                                          ←Timer A3 interrupt
        .word   dummy_int
TA2I:                                          ←Timer A2 interrupt
        .word   dummy_int
TA1I:                                          ←Timer A1 interrupt
        .word   dummy_int
TA0I:                                          ←Timer A0 interrupt
        .word   dummy_int
INT2:                                          ←External interrupt INT2
        .word   dummy_int
INT1:                                          ←External interrupt INT1
        .word   dummy_int
INT0:                                          ←External interrupt INT0
        .word   dummy_int
WDT:                                           ←Watchdog timer interrupt
        .word   dummy_int
RESERVED:                                      ←Debugger interrupt
        .word   dummy_int
BRK:                                           ←BRK instruction
        .word   dummy_int
DIV0:                                          ←Division by 0 interrupt
        .word   dummy_int
;
RESET:                                         ←Reset
        .word   offset start
```

* dummy_int is a dummy interrupt processing function.

Figure 2.28 Interrupt Vector Address Table (section.inc)

The contents of the interrupt vectors varies according to the machine in the 7700 series. See the User Manual for your machine for details.

Change the interrupt vector address table as follows:

[1] Externally declare the interrupt processing function in the .RXTrasm77 pseudo instruction. The labels of functions created by NC77 are preceded by the underscore (_). Therefore, the names of interrupt processing functions declared here should also be preceded by the underscore.

[2] Replace the names of the interrupt processing functions with the names of interrupt processing functions that use the dummy interrupt function name dummy_int corresponding to the appropriate interrupt table in the vector address table.

Figure 2.27 is an example of registering the UART1 send interrupt processing function uarttrn.

```
        .section        vector                  ; Interrupt vector table
        .ext    _uarttrn                                 ⇐Process [1] above
        .org    0ffd6H
ADCOMP:
        .word   dummy_int
TRN1:
        .word   _uarttrn                                 ⇐Process [2] above
REC1:
        :
    (remainder omitted)
```

Figure 2.29 Example Setting of Interrupt Vector Addresses (section.inc)

Chapter 3
Programming Technique

This chapter describes how to use integers and variables in your programs and how to specify nc77 command line options so that the code generated by NC77 is more efficient.

3.1 Notes

3.1.1 Notes about Version-up

The machine-language instructions (assembly language) generated by NC77 vary in contents depending on the startup options specified when compiling, contents of version-up, etc. Therefore, when you have changed the startup options or upgraded the compiler version, be sure to reevaluate the operation of your application program.

Furthermore, when the same RAM data is referenced (and its contents changed) between interrupt handling and non-interrupt handling routines or between tasks under realtime OS, always be sure to use exclusive control such as volatile specification. Also, use exclusive control for bit field structures which have different member names but are mapped into the same RAM.

3.1.2 Optimization

a. Suppressing Optimization

In NC77, the code shown in Figure 3.1 would be optimized by default, even without the option -O.

```
extern int port;

func()
{
    port;
}
```

Figure 3.1 Example of Optimizing Code

In this example, the code has been written to read the port. However, if this code is optimized, no code is output. To suppress optimization, add the volatile modifier, as shown in Figure 3.1.

```
extern int volatile port;

funC()
{
    port;
}
```

Figure 3.2 Example of Suppressing Optimization

b. Code Generation

In NC77, the code shown in Figure 3.3 would be optimized by default, even without the option -O.

```
int func(char c)
{
    int i;

    if(c != -1)
        i = 1;
    else
        i = 0;
    return i;
}
```

Figure 3.3 Example of Optimizing Code

In this example, variable c takes the char type and is therefore handled by NC77 as an unsigned char type. Because the range of values that can be represented by unsigned char types is 0 to 255, variable c cannot take the value -1.

Therefore, be aware that NC77 will not generate any assembler code for similar statements that are logically not possible.

3.1.3 Using the Register Variables

a. Enabling the register Modifier

When the register modifier is specified for local variables in a function, option -fenable_register (-fER) must be specified. If it is not specified, the variables for which the register modifier was specified will be processed as auto variables.

b. Optimization of register Variables

When parameters are passed to a function via the registers, those parameters are temporarily moved to the auto area (stack frame).When you specify option -O, -OR, or -OS, parameters passed via the registers may, to improve code efficiency, be processed as register variables rather than being moved to the auto area.

3.2 Greater Code Efficiency

3.2.1 Programming Techniques for Greater Code Efficiency

a. Regarding Integers and Variables

[1]Unless required, use unsigned integers. If there is no sign specifier for int, short, or long types, they are processed as signed integers. Unless required, add the 'unsigned' sign specifier for operations on integers with these data types.*1

[2]If possible, do not use >= or ⇐ for comparing signed variables. Use != and == for conditional judgements.

b. far type array

The far type array is referenced differently at machine language level depending on its size.

[1]When the array size is within 64 Kbytes
Subscripts are calculated in 16-bit width. This ensures efficient access for arrays of 64 Kbytes or less in size.

[2]When the array size is greater than 64 Kbytes or unknown
Subscripts are calculated in 32-bit width.

Therefore, when it is known that the array size does not exceed 64 Kbytes, explicitly state the size in extern declaration of far type array as shown in Figure 3.4 or add the -fsmall_array (-fSA)*2 option before compiling. This helps to increase the code efficiency of the program.

```
extern int far array[]; ⇐Size is unknown, so subscripts are calculated as 32-bit values.

extern int far array[10]; ⇐Size is within 64KB, so access is more efficient.
```

Figure 3.4 Example extern-Declaration of far Array

*1. If there is no sign specifier for char-type or bitfield structure members, they are processed as unsigned.

*2. When the -fsmall_array (-fSA) option is specified, the compiler assumes an array of an unknown size to be within 64 Kbytes as it generates code.

c. Array Subscripts

Array subscripts are type-extended during operations according to the size of each element in the array.

[1]2 bytes or more (other than char or signed char types)
Subscripts are always extended to int types for operations.
[2]far arrays of 64KB or more
Subscripts are always extended to long types for operations.

Therefore, if you declare variables that will be array subscripts as char types, they will be extended to int types each time they are referenced and therefore the code will not be efficient. In such cases, declare variables that will be array subscripts as int types.

d. Using Prototype declaration Efficiently

NC77 allows you to accomplish an efficient function call by declaring the prototype of a function.

This means that unless a function is declared of its prototype in NC77, arguments of that function are placed on the stack following the rules listed in Table 3.1 when calling the function.

Table 3.1 Rules for Using Stack for Parameters

Data type(s)	Rules for pushing onto stack
char signed char	Expanded into the int type when stacked.
float	Expanded into the double type when stacked.
otherwise.	Not expanded when stacked.

For this reason, NC77 may require redundant type expansion unless you declare the prototype of a function.

Prototype declaration of functions helps to suppress such redundant type expansion and also makes it possible to assign arguments to registers. All this allows you to accomplish an efficient function call.

e. nc77 Command Line Options

nc77 command line options include those for optimizing speed and ROM efficiency, and those for selecting branch instructions.

1. Optimization options for speed and ROM efficiency
   - -O .... Maximum optimization of speed and ROM size
   - -OR .... Maximum optimization of ROM size followed by speed
   - -OS .... Maximum optimization of speed followed by ROM size
2. Options for selection of branch instructions
   - -OB1 .... Generates branch instructions taking only ROM size into account (default)
   - -OB2 .... Generates branch instructions taking speed into account (in same bank)
   - -OB3 .... Generates branch instructions taking speed into account (outside bank)

By specifying combinations of these options on the nc77 command line, you can control how code is generated to optimize memory efficiency and speed.

For example, specifying -OB2 or -OB3 with -OS for a program that contains a function that is mapped across a bank boundary prevents the BRA instruction from being generated, as shown in Table 3.2. With this combination of options, the generated code is optimized more for speed.

Table 3.2 Combinations of Branch Instruction Selection Option and -OS Option

Option	Instruction generated when combined with -OS option
-OB2	JMP
-OB3	JMPL

To focus on ROM capacity, you can generated efficient code by combining the above six optimization options with options for modifying the generated code.

For example, if both program and data are within 64KB and are in the same bank, you can specify one of the options shown below to generate code that optimizes both ROM size and speed.

- -fnear_function ..... Processes all functions without near or far attributes as having the near attribute
- -OS .... Optimizes more for speed than ROM size
- -OB2 .... Optimizes branch instructions for the same bank focussing more on speed

f. Techniques for Controlling near and far Attributes of Functions

You can control whether JSR or JSRL is used to call a function by specifying near or far.

Normally, if a program exceeds 64KB, the functions are all processed as far functions. However, to minimize ROM size and stack size, you can use near functions as necessary.

Specifying near or far for functions is different from specifying near or far for data. In the case of functions, you specify near or far according to whether the function is in the bank currently indicated by the program bank register (PG).

That is, there is no bank specified for near functions, but if a function is in a bank shown by the current value of the PG, it is a near function.. If you selectively define near functions, the calling program bank and the definition program bank must be the same.

Thus, provided the functions in a file are in the same section, you can use the fact that there is a high likelihood of them being mapped to the same program bank to make only static functions near functions.

To do so, specify the command line option -ffar_program_section (-fFPS) when compiling to map both near and far functions to the same section. This method allows you to reduce the risk of errors when linking even if you only selectively define near functions.

g. Optimizing Speed of Getting 32-bit Results From 16-bit Multiplication Operations

By casting 16-bit data as long types, NC77 will generate code to obtain 32-bit results without extending the data to 32-bit types.

```
long  func(int  i1,int  i2)
{
    long  l1,l2,l3;

    l1  =  i1;
    l2  =  i2;
    l3  =  i3;
    l3  =  l1  *  l2;
    return  i3;
}
```

Figure 3.5 Example of Obtaining 32-bit Results From Multiplication Operations on 16-bit Data (1)

The example in Figure 3.5 is written so that the result of multiplying 16-bit variables i1 and i2 is stored in 32-bit variable l3. However, because the 16-bit data is assigned to a 32-bit variable, there is a loss of code efficiency.

In such cases, you can use the cast operator, as shown in Figure 3.6, to improve the code efficiency.

```
long  func(int  i1,int  i2)
{
    long  l;

    l  =  (long)i1  *  (long)i2;
    return  l;
}
```

Figure 3.6 Example of Obtaining 32-bit Results From Multiplication Operations on 16-bit Data (2)

h. Other methods

In addition to the above,the ROM capacity can be compressed by changing program description s as shown below.

(1) Chabge a relatively small function that is called only once to an inline function.
(2) Replace an if-else statement with a switch statement. (This is effective unless the variable concerned is a simple variable such as an array,pointer,or structure.)
(3) For bit comparison, use '&' or '|' in place of '&&' or '||'.
(4) For a function which returns a value in only the range of char type, declare its return value type with char.
(5) For variables used overlapping a function call, do not use a register variable.

3.2.2 Speeding Up Startup Processing

The section.inc startup program includes routines for clearing the bss area. This routine ensures that variables that are not initialized have an initial value of 0, as per the C language specifications.

For example, the code shown in Figure 3.7 does not initialize the variable, which must therefore be initialized to 0 (by clearing the bss area) during the startup routine.

```
static int i;
```

Figure 3.7 Example Declaration of Variable Without Initial Value

In some instances, it is not necessary for a variable with no initial value to be cleared to 0. In such cases, you can comment out the routine for clearing the bss area in the startup program to increase the speed of startup processing.

```
;====================================================================
; NEAR area initialize.
;--------------------------------------------------------------------
; bss_NE & bss_NO zero clear
;--------------------------------------------------------------------
;   BZERO bss_NEsz,bss_NE_top
;   BZERO bss_NOsz,bss_NO_top
                :
            (omitted)
                :
;====================================================================
; FAR area initialize.
;---------------------------------------------------------------------
; bss_FE & bss_FO zero clear
;---------------------------------------------------------------------
;   BZERO bss_FEsz,bss_FE_top
;   BZERO bss_FOsz,bss_FO_top
                :
             (omitted)
                :
```

Figure 3.8 Commenting Out Routine to Clear bss Area

3.3 Linking Assembly Language Programs with C Programs

3.3.1 Calling Assembler Functions from C Programs

a. Calling Assembler Functions

Assembler functions are called from C programs using the name of the assembler function in the same way that functions written in C would be.

The first label in an assembler function must be preceded by an underscore (_). However, when calling the assembly function from the C program, the underscore is omitted.

The calling C program must include a prototype declaration for the assembler function.

Figure 3.9 is an example of calling assembler function asm_func. In this example, the prototype declaration for asm_func has the near attribute, and the function is therefore called with JSR.

```
extern void near        asm_func( void );          ⇐Assembler function
                                                     prototype declaration
void main()
{
                :
                 (omitted)
                :
        asm_func();          ⇐Calls assembler function
}
```

Figure 3.9 Example of Calling Assembler Function Without Parameters(smp1.c)

```
       .pub     _main
_main:
                :
                 (omitted)
                :
        jsr     _asm_func     ⇐Calls assembler function(preceded by '_')

        rtl
```

Figure 3.10 Compiled result of smp1.c(smp1.a77)

* 1.The instruction for calling the assembler function differs according to whether near or far is included in the prototype declaration. JSR is used to call near functions, while JSRL is used to call far functions. Similarly, RTS must be used to return from a near assembler function to the calling C program, while RTL must be used to return from a far function.

b. When assigning arguments to assembler functions

When passing arguments to assembler functions, use the extended function "#pragma PARAMETER." This #pragma PARAMETER passes arguments to assembler functions via pair of 16-bit register (AB, XY), 16-bit registers (A, B, X, Y), or 8-bit registers (A, B, X, Y).

The following shows the sequence of operations for calling an assembler function using #pragma PARAMETER:

[1]Write a prototype declaration for the assembler function before the #pragma PARAMETER declaration. You must also declare the parameter type(s).
[2]Declare the name of the register used by #pragma PARAMETER in the assembler function's parameter list.

Figure 3.11 is an example of using #pragma PARAMETER when calling the assembler function asm_func.

```
extern unsigned int        asm_func(unsigned int, unsigned int);
#pragma PARAMETER       asm_func(X, Y)      ⇐Parameters are passed via the
                                              X  and Y registers to the
void main()                                   assembler function.
{
        int         i = 0x02;
        int         j = 0x05;

        asm_func(i, j);                      ⇐Calling assembler function
}
```

Figure 3.11 Example of Calling Assembler Function With Parameters (smp2.c)

```
    .cline      6
;## # C_SRC :           int         i = 0x02;
    ldm.W #0002H,DP:3 ;  i
    .cline      7
;## # C_SRC :           int         j = 0x05;
    ldm.W #0005H,DP:1 ;  j
    .cline      9
;## # C_SRC :           asm_func(i, j); ⇐Parameters are passed via the X  and Y
    ldy   DP:1  ;  j                      registers to the assembler function.
    ldx   DP:3  ;  i
    jsrl  _asm_func               ⇐Calls assembler function(preceded by '_')
```

Figure 3.12 Compiled result of smp2.c(smp2.a77)

c. Limits on Parameters in #pragma PARAMETER Declaration

The following parameter types cannot be declared in a #pragma PARAMETER declaration.

- structure types and union type parameters
- Floating point type (float and double) parameters

3.3.2 Writing Assembler Functions

a. Writing Called Assembler Functions

The following describes how to write entry processing for assembler functions:

1. Specify the section name with the assembler pseudo instruction .SECTION. Sections can take any name.
2. Specify the start of the function using the assembler pseudo instruction .FUNC.
3. Specify the function name label using the assembler pseudo instruction .PUB as public.
4. Precede the function name with the underscore (_) for use as a label.
5. If the value of the DT and DPR registers are changed, save them to the stack.

The data size selection flag (m), index register size selection flag (x), and decimal operation mode flag (D) are all cleared when a function is called from a C program. Also clear these flags before returning from the function to the C program.

The following describes how to write exit processing for assembler functions:

6. If the value of the DT and DPR registers save them to the stack, return them from the stack.
7. Clear the m, x, and D flags.
8. Specify instruction RTS or RTL.
9. Specify the function name label at the end of the function using the RASM77 pseudo instruction .ENDFUNC.

Figure 3.13 is an example of how to code an assembler function. In this example, the section name is program_N, which is the same as the section name output by NC77.

```
    .SECTION        program_N       ⇐[1]
    .FUNC   _asm_func               ⇐[2]
    .PUB    _asm_func               ⇐[3]
_asm_func:                          ⇐[4]
    PHD                             ⇐[5]
    LDT     #10H
    STA     A, DT:MEMO1
    STA     B, DT:MEMO2
    SEM
    LDA.B A, #7H
                :
            (abbreviated)
                :
    PLT                             ⇐[6]
    CLP     m,x,D                   ⇐[7]
    RTS                             ⇐[8]
    .ENDFUNC                        ⇐[9]
```

* [1] to [9] correspond to the steps described above.

Figure 3.13 Example Coding of Assembler Function

b. Returning Return Values from Assembler Functions

When returning values from an assembler function to a C language program, registers can be used through which to return the values for the integer, pointer, and floating-point(only float type) types. Table 3.3 lists the rules on calls regarding return values. Figure 3.14 shows an example of how to write an assembler function to return a value.

Table 3.3 Calling Rules for Return Values

Return value type	Rules
char type	A register
int type near pointer type	A register
float type long type far pointer type	The 16 low-order bits are stored in the A register and the 16 high-order bits are stored in the B register as the value is returned.
double type long double type Compound type	Immediately before calling the function, the far address indicating the area for storing the return value is pushed to the stack. Before the return to the calling program, the called function writes the return value to the area indicated by the far address pushed to the stack.

```
          .SECTION         program
          .FUC             _asm_func
          .PUB             _asm_func
_asm_func:
                :
                 (omitted)
                :
          LDA.W A, #1A00H   ;Low 16 bits of 32-bit data
          LDA.W B, #0000H   ;High 16 bits of 32-bit data
          CLP   m, x, D
          RTS
          .ENDFUNC
          .END
```

Figure 3.14 Example of Coding Assembler Function to Return long-type Return Value

c. Referencing C Variables

Because assembler functions are written in different files from the C program, only the C global variables can be referenced.

When including the names of C variables in an assembler function, precede them with an underscore (_). Also, in assembler language programs, external variables must be declared using the assembler pseudo instruction .EXT.

Figure 3.15 is an example of referencing the C program's global variable counter from the assembler function asm_func.

[C program]

```
unsigned int    counter;            ⇐C program global variable

main()
{
                :
            (omitted)
                :
}
```

[Assembler function]

```
        .EXT            _counter    ⇐External declaration of C program's
_asm_func:                            global variable
                :
            (omitted)
                :
        LDA.W           A, _counter ⇐Reference
```

Figure 3.14 Referencing a C Global Variable

d. Notes on Coding Interrupt Handling in Assembler Function

If you are writing a program (function) for interrupt processing, the following processing must be performed at the entry and exit.

1. Save the registers (A, B, X, Y, DPR and DT) at the entry point.
2. Restore the registers (A, B, X, Y, DPR and DT) at the exit point.
3. Use the RTI instruction to return from the function.

Before saving or restoring the registers, always make sure that the data size selection flag (m) and index register size flag (x) are the same.

- When using DT-dependent addressing mode instructions in an interrupt processing program, remember to load the value of the DT register at the entry to the interrupt processing program.[*1]

Figure 3.16 is an example of coding an assembler function for interrupt processing.

* 1. The code generated by the C compiler may temporarily change the DT register. Therefore, if your assembly language interrupt processing program "uses DT-dependent addressing modes" , save the DT value at the beginning of the interrupt processing program, then load the required DT value.

```
     .SECTION        interrupt
     .DT       __DT
     .FUNC     _int_func
     .PUB      _int_func
_int_func:
     CLP       m,x                  ←Clear m and x flags.            ┐
     PSH       A,B,X,Y,DPR,DT       ←Push registers.                 │
     LDT       #__DT                ←Set the DT value to be          │ Save and restore
     SEM                              used in the interrupt          │ with the m and x
     LDA.B     A, #7H                 processing program             │ flags in the same
                   :                                                 │ state.
              (abbreviated)                                          │
                   :                                                 │
     CLP       m,x                  ←Clear m and x flags.            │
     PUL       A,B,X,Y,DPR,DT       ←Pull registers.                 │
     RTI                            ←Return to C program             ┘
     .ENDFUNC
     .END
```

Figure 3.16 Example Coding of Interrupt Processing Assembler Function

e. Notes on Calling C Functions from Assembler Functions

Note the following when calling a function written in C from an assembly language program.

(1) Call the C function using a label preceded by the underscore (_) or the question mark(?).
(2) Clear the m, x, and D flags before calling the C function.
(3) Use JSR to call functions with the near attribute and JSRL to call functions with the far attribute.
(4) Call the C function only after loading the data bank register (DT) value specified when compiling it. If not value is specified when the function is compiled, call the function after loading a bank value of 0 into the DT.

3.3.3 Notes on Coding Assembler Functions

Note the following when writing assembly language functions (subroutines) that are called from a C program.

a. Notes on Handling m, x and D flags

The m, x, and D flags in the processor status register must all be cleared before the function is called from the C program. They must also be cleared before when returning from the function to the C program.

b. Notes on Handling DT and DPR Register

If the values of the DPR (direct page register) or DT (data bank register) are changed by the assembler function, you will be unable to make a normal return to the C program from which the function was called. It is therefore important that you do not change these values in the function. If, because of the system's design, it is unavoidable, save the values to the stack at the start of the function, then restore them before returning to the C program.

c. Notes on Handling A, B, X and Y Registers

No problem arises if the contents of the A, B, X, and Y registers are changed by an assembler function.

d. Passing Parameters to an Assembler Function

Use the #pragma PARAMETER function if you need to pass parameters to a function written in assembly language. The parameters are passed via registers. Figure 3.17 shows the format (asm_func in the figure is the name of an assembler function).

```
unsigned int near asm_func(unsigned int, unsigned int);
                                        ↑Prototype declaration of assembler function

#pragma PARAMETER asm_func(A,B)
```

Figure 3.17 Example Coding of Assembler Function

#pragma PARAMETER passes arguments to assembler functions via 16-bit registers (A, B, X, Y) and 8-bit registers (A, B, X, Y). In addition, the 16-bit are combined to form 32-bit registers (AB and XY) for the parameters to be passed to the function. Note that an assembler function's prototype must always be declared before the #pragma PARAMETER declaration.

However, you cannot declare the following parameter types in a #pragma PARAMETER declaration:

- struct or union types
- floating point type(double) argument

You also cannot declare the functions returning structure or union types as the function's return values.

3.4 Other

3.4.1 Precautions on Transporting between NC-Series Compilers

NC77 basically is compatible with Mitsubishi C compilers "NCxxx" at the language specification level (including extended functions). However, there are some differences between the compiler (this manual) and other NC-series compilers as described below.

a. Difference in default near/far

The default near/far in the NC series are shown in Table 3.5. Therefore, when transporting the compiler (this manual) to other NC-series compilers, the near/far specification needs to be adjusted.

Table 3.5 Default near/far in the NC Series

Compiler	RAM data	ROM data	Program
NC308	near (However, pointer type is far)	far	far Fixed
NC30	near	far	far Fixed
NC79	near	near	far
NC77	near	near	far

3.4.2 7700 Family-Dependent Code

Some hardware specifications differ according to the model in the 7700 family. The following points should therefore be noted.

(1)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.

(2)In the M37700/M37701 group, you must specify command line option -OB2 or -OB3 when the program is mapped to other than bank 0.

(3)Make sure that the RTS, JMP, and JSR 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 warning option (-C) when linking. This option causes a warning message to be displayed if RTS, JMP, or JSR are mapped to a bank boundary.

(4)Some models have a function whereby the stack area is mapped to the last bank (bank 255). However, this function cannot be used with NC77.

3.4.3 General Notes on Porting

When you upgrade your C compiler to a new version, the generated machine language (assembly language) may also change. Therefore, be sure to check the assembly language generated by the new version of the C compiler for the following code:

1.Processing-speed dependent code
2.Generated assembly language-dependent code

When porting from C77 V.2.10 or earlier or MR7700 V.2.12 or earlier, there are many incompatibilities at the C source level. It is therefore essential to leave sufficient time for making the transition from these versions.

3.4.4 Porting from C77 V.2.10 or Earlier

a. Language Specifications

(1)Version 3.20 of NC77 allows you to perform signed division and right shifts, which were not available in C77 V.2.10 and earlier. Therefore, if your program uses such calculation expressions, the results will be different in NC77 V.5.xx from those in C77 V.2.10 or earlier.

```
int         i, j;

i = -2;
j = i >> 1;
```

Figure 3.18 Example Coding of Signed Operations

(2)In NC77 V.5.xx, multidimensional arrays are addressed according to ANSI rules, whereas proprietary specifications were used in C77 V.2.10 and earlier. Therefore, you may get different results in the respective versions if you are performing addition or subtraction on the addresses of multidimensional arrays.

(3)When the sizeof operator is used on a character string, the size of the starting address of the area storing the character string was returned in C77 V.2.10 and earlier. In NC77 V.5.xx, however, the size of the area storing the character string is returned.

```
sizeof("NC77");
```

Return value	NC77 V.5.xx	C77 V.2.10 and earlier
sizeof("NC77"); return value	5	Small model: 2
		Large model: 4

Figure 3.19 Comparison of Results of sizeof Operation

(4)Because there was no support for floating point types in C77 V.2.10 and earlier, float types and long double types were all processed as long int types. Floating point types are, however, supported in NC77 V.5.xx. Therefore, because the 1.0 in the code in Figure 3.20 is double type, the operation on the right is performed as a double-type operation. As a result, there is a possible loss of code efficiency and of execution speed.

```
i = j + 1.0;
```

Figure 3.20 Example Coding of Floating Point Operation

(5)In C77 V.2.10 and earlier, a 1-byte dummy was inserted when mapping structures if there were 1-byte members. In NC77, no dummy is inserted unless #pragma STRUCT is specified, and packing is therefore applied to the mapped structures. As a result, structures are mapped to different locations in C77 V.2.10 and earlier and NC77. Application programs that perform operations after casting structure pointers to char * types, etc., will therefore not run correctly.

(6)In C77 V.2.10 and earlier, no memory is allocated to variable i in code such as that shown in Figure 3.21. In NC77, memory is allocated as per the standard C specifications.Therefore, when porting a program written using V.2.10 or earlier to NC77, duplicate definition errors may occur when the program is linked. If these errors occur, modify the code to satisfy the standard C specifications.

```
extern int i;

int i = 0;
```

Figure 3.21 Example extern Declaration of Variable

(7)In C77 V.2.10 and earlier, some items that should cause errors did not. In the source program shown in Figure 3.22, for example, a duplicate definition error should be output for function gf, but this did not happen in C77 V.2.10 and earlier. However, errors are output in NC77 V.5.xx and you should therefore remember to perform a prototype declaration for that function.

```
func()
{
        unsigned int i;
        i = gf();
}

unsigned int gf()
{
        return  0;
}
```

Figure 3.22 Example Code Resulting in Output of Duplicate Function Definition Error

b. Interfacing to Assembler Functions

The method of storing the return values of the following functions differs in NC77 V.5.xx and C77 V.2.10. You must therefore modify the assembler functions equivalent to these functions.

- Functions returning structures
- Functions returning double types

c. Using the asm Function

(1)Modify any code that uses the CLM instruction, etc., to change the data size selection flag (m) and index register size flag (x) to match the code shown in Figure 3.23.

```
        asm(MFLAG, XFLAG);

MFLAG: Status of data size selection flag
XFLAG: Status of index register size selection flag
Status: 0 ........... Flag cleared
        1 ........... Flag set
        2 ........... Do not switch flag
```

Figure 3.23 Format for Switching m and x Flags

(2)When specifying storage class auto variables or parameters using the offset of the direct page register (DP), modify the code as shown in Figure 3.24.

```
        asm( "    operation code    A,DP:$$" , name_of_auto_variable);
```

Figure 3.24 Format to Specify DT Offset

d. #pragma EQU Compatibility

In NC77 V.5.xx, variables with absolute addresses are compiled as if the area of that variable is already secured at that address. Therefore, extern declarations and static declarations for that variable are ignored. You also cannot change the symbol for the absolute address to a global symbol.

Use the following procedure if, when linking with NC77 V.5.xx, a symbol specified in #pragma EQU cannot be resolved.

(1)Error in C programs

Create a header file containing all #pragma EQU declarations and include that header file at the start of each program.

(2)Error in assembler function

When calling #pragma EQU variables declared in a C program from an assembler function, specify all called variables in the assembler function in the pseudo instruction .EQU. You can easily create a group of such pseudo instructions using the following method.

1.Declare the called variables in a C header file and compile it using NC77 V.5.xx with the -S option to create an assembly language source file.

2.The assembly language source file will contain a group of .EQU pseudo instructions which can be copied to the beginning of the assembly language source file.

e. Using Programs Compiled with C77 V.2.10 or Earlier

The format for calling functions differs in NC77 V.5.00 from C77 V.2.10 and earlier. Therefore, libraries and object files compiled using C77 V.2.10 or earlier will not run correctly when linked with libraries and object files compiled with NC77 V.5.xx. You must therefore recompile all libraries and object files using NC77 V.5.xx.

f. Using Interrupt Processing Functions Declared in #pragma INTF

Declare the return values and parameters of interrupt processing functions declared in #pragma INTF as void types.

```
#pragma INTF      intfunc
intfunc()
{
                  :
         (abbreviated)
                  :
}

        ▼

#pragma INTF      intfunc
void  func(void)
{
                  :
         (abbreviated)
                  :
}
```

Figure 3.25   Example Modification of Interrupt Processing Function

g. Standard I/O Library Functions

In NC77 V.5.xx, the pointer variables used in the following standard I/O library functions are compiled as near types in the supplied libraries.

Table 3.6   Standard I/O Library Functions

Function	Function
fgetc	fputs
getc	puts
fgets	fwrite
gets	printf
fread	fprintf
scanf	sprintf
fscanf	ungetc
sscanf	ferror
fputc	feof
putc	

If you are using the former large model such that pointer variables are processed as far types, use the make file (make.far, or makefar.dos in the MS-DOS version) in the directory containing the standard library function source file to remake the library file.

h. peek and poke Library Functions

In NC77 V.5.xx, you can now use the far pointer to access any part of the whole of the 7700 family memory space. The peek and poke library functions used in C77 V.2.10 and earlier have therefore been deleted.

Change peek and poke library functions to memcpy or bcompy library functions, etc.

i. divr and modr Library Functions

In NC77 V.5.xx, you can now perform signed division. The divr and modr library functions used in C77 V.2.10 and earlier have therefore been deleted.

Modify the code to use / and %.

j. Abolition of -Za Option and Modification of Handling char-type Parameters

In C77 V.2.10 and earlier, the -Za option was used to control whether the parameters used when calling functions with char-type parameters were loaded as 8-bit or 16-bit parameters. However, the -Za option has been abolished from NC77 and the existence of a prototype declaration for the function's parameters now control whether char-type parameters are loaded as 8-bit or 16-bit parameters. [*1]

Include prototype declarations (to declare the type of parameters used by the function) for functions with char-type parameters in both the file that uses the function and the file that defines the function.

k. Prototype Declarations

In NC77 V.5.xx, the existence of a prototype declaration determines how parameters are saved to the stack when a function is called. Therefore, if you have a program created using C77 V.2.10 or earlier and that program includes prototype declarations, it will not run properly under NC77 V.5.xx unless there are prototype declarations in all files that call and all files that define the function for which the prototype declaration exists. You must therefore make sure that there are prototype declarations in all files that call or define the function.

l. Section Names

Section names output by NC77 V.5.xx are not the same as those output by C77 V.2.10 or earlier. Therefore, you must modify the section names in assembly language programs that use section names output by the compiler.

3.4.5 Porting from NC77 V.3.00

a. The -fext_const_set_rom_section (-fECSRS) Option

The nc77 command line option -fext_const_set_rom_section (-fECSRS) is now the default. To compile using the same default as previous versions, specify the nc77 command line option -fext_const_unset_rom_section (-fECURS) when compiling.

b. Memory Management Library Functions

In NC77 V.3.00 the following memory management library functions secured and released memory in the near area. In NC77 V.3.10, however, memory is secured and released in the far area. You must therefore change the pointers used by the memory management library functions to far pointers.

* 1. This modification is to ensure conformity to ANSI standards.

Table 3.7 Memory Management Library Functions

Function
calloc
free
malloc
realloc

3.4.6 Porting from MR7700 V.2.12 or Earlier

When porting application programs developed using version 2.12 or earlier of the MR7700 realtime operating system, use the new NC77 V.5.xx functions #pragma TASK and #pragma INTHANDLER to specify tasks and handlers. These functions automatically generate the code for the entry and exit processing of tasks and interrupt handlers.

1. Specify #pragma TASK for tasks.
2. By specifying interrupt handlers in #pragma INTHANDLER , you no longer have to call the IntEntry macro or ret_int system call. These must be deleted from your files.

Figure 3.26 shows examples of how to make the required modifications for the above two points. (tas in the figure is the task name; hand is the interrupt handler name.)

```
#pragma INTF hand

tas()
{
            :
        (abbreviated)
            :
    ext_tsk();
}
hand()
{
    IntEntry();
            :
        (abbreviated)
            :
    ret_int();
}

    ▼

#pragama TASK tas
#pragama INTHANDLER hand

void tas()
{
            :
        (abbreviated)
            :
/*      ext_tsk(); */                  ←Comment out or delete
}
void hand()
{
/*    IntEntry(); */                   ←Comment out or delete
            :
        (abbreviated)
            :
/*    ret_int();  */                   ←Comment out or delete
}
```

Figure 3.26 Example Modification of Task and Interrupt Handler Format

3.The ret_wup system call, which performs returns from interrupt handlers and activates tasks, performs the same operations as the ret_int system call and also activates the specified task. If the interrupt handler is specified in #pragma INTHANDLER, the ret_int-equivalent code is automatically generated, so ret_wup should be changed to the iwup_tsk system call, which activates the specified task. Figure 3.27 is an example of how to change the ret_wup system call. (tas in the figure is the task name; hand is the interrupt handler name.)

```
#pragma INTF hand()

hand()
{
        IntEntry();
                  :
              (abbreviated)
                  :
        ret_wup(ID_tas);
}

        ▼

#pragama TASK tas
#pragama INTHANDLER hand

void hand(void)
{
/*      IntEntry(); */                          ←Comment out or delete
                  :
              (abbreviated)
                  :
      iwup_tsk(ID_tas);                         ←Change to iwup_tsk system call
}
```

Figure 3.27 Example Modification of ret_wup System Call Format

Appendix A
Command Option Reference

This appendix describes how to start the compile driver nc77 and the command line options. The description of the command line options includes those for the rasm77 assembler and link77 linkage editor, which can be started from nc77.

A.1 nc77 Command Format

```
%    nc77Δ[command-line-option]Δ[assembly-language-source-file-name]Δ
       [relocatable-object-file-name]Δ<C-source-file-name>

%        : Prompt
< >      : Mandatory item
[ ]      : Optional item
Δ        : Space
```

Figure A.1 nc77 Command Line Format

```
% nc77 -osample -rasm77 "-l" -link77 "-ms" ncrt0.a77 sample.c<RET>

<RET> : Return key
 * Always specify the startup program first when linking.
```

Figure A.2 Example nc77 Command Line

A.2 nc77 Command Line Options

A.2.1 Options for Controlling Compile Driver

Table A.1 shows the command line options for controlling the compile driver.

Table A.1 Options for Controlling Compile Driver

Option	Function
-c	Creates a relocatable file (extension .r77) and ends processing [*1]
-D*identifier*	Defines an identifier. Same function as #define.
-I*directory*	Specifies the directory containing the file(s) specified in #include. You can specify up to 8 directories.
-E	Invokes only preprocess commands and outputs result to standard output.[*1]
-P	Invokes only preprocess commands and creates a file(extension .i). [*1]
-S	Creates an assembly language source file (extension .a77) and ends processing.[*1]
-U *predefined macro*	Undefines the specified predefined macro.
-silent	Suppresses the copyright message display at startup.

1. If you do not specify command line options -c, -E, -P, or -S, nc77 finishes at and output files up to the machine language data file (extension .hex) are created.

-c

Compile driver control

Function : Creates a relocatable object file (extension .r77) and finishes processing

Execution example :

```
%nc77 -c sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

sample.c

% ls sample.*
-rw-r--r--  1 toolusr      2835 Aug 17 11:28 sample.c
-rw-r-----  1 toolusr       450 Aug 17 11:28 sample.r77
%
```

Notes : If this option is specified, no machine language data file (extension .hex) or other file output by link77 is created.

-D*identifier*

Compile driver control

Function : The function is the same as the preprocess command #define. Delimit multiple identifiers with spaces.

Syntax : nc77Δ-D*identifier*[=*constant*]Δ<C source file>

[= *constant*] is optional.

Notes : The number of identifiers that can be defined may be limited by the maximum number of characters that can be specified on the command line of the operating system of the host machine.

-I*directory*

Compile driver control

Function : Specifies the directory containing the files specified in the #include preprocess command. You can specify up to 8 directories.

Syntax : nc77Δ-I *directory*Δ<C source file>

Execution example :

```
% nc77 -c -I./test/include -I./test/inc sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

sample.c

%
```

* In this example, two directories, ./test/include and ./test/inc are specified.

Notes : The number of directories that can be defined may be limited by the maximum number of characters that can be specified on the command line of the operating system of the host machine.

-E

Compile driver control

Function : Invokes only preprocess commands and outputs results to standard output

Execution example :

```
% nc77 -E sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

#line 1 "sample.c"
          :
      (omitted)
          :
#line 1 "/usr3/tool/toolusr/work30/inc77/stdio.h"
          :
      (omitted)
          :
```

Notes : When this option is specified, no assembly source file (extensions .a77), relocatable object files (extension .r77), machine language data files (extension .hex), or other files output by ccom77, rasm77, or link77 are generated.

-P

Compile driver control

Function : Invokes only preprocess commands and creates a file (extension .i)

Execution example :

```
% nc77 -P sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

sample.c
%ls sample.*
-rw-r--r--  1 toolusr      2835 Aug 17 11:28 sample.c
-rw-r-----  1 toolusr      2322 Aug 17 11:30 sample.i
%
```

Notes :

1. When this option is specified, no assembly source file (extensions .a77), relocatable object files (extension .r77), machine language data files (extension .hex) or other files output by ccom77, rasm77, or link77 are generated.
2. The file (extension .i) generated by this option does not include the #line command generated by the preprocessor. To get a result that includes #line, try again with the -E option.

-S

Compile driver control

Function : Creates assembly language source files (extension .a77 and .ext) and stops processing

Execution example :

```
% nc77 -S sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

sample.c
% ls sample.*
-rw-r-----  1 toolusr      2059 Aug 17 11:30 sample.a77
-rw-r--r--  1 toolusr      2835 Aug 17 11:28 sample.c
%
```

Notes : When this option is specified, no relocatable object files (extension.r77), machine language data files (extension .hex) or other files output by rasm77 or link77 are generated.

-U*predefined macro*

Compile driver control

Function : Undefines predefined macro constants

Syntax : nc77Δ-U *predefined macro*Δ<C source file>

Execution example :

```
% nc77 -c -UNC77 -UMELPS sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

sample.c

%
```

*In this example, macro definitions NC77 and MELPS are undefined.

Notes : The maximum number of macros that can be undefined may be limited by the maximum number of characters that can be specified on the command line of the operating system of the host machine.
STDC, _LINE_, _FILE_, _DATE_, and _TIME_ cannot be undefined.

-silent

Compile driver control

Function : Suppresses the display of copyright notices at startup

Execution example :

```
% nc77 -c -silent sample.c
sample.c

%
```

A.2.2 Options Specifying Output Files

Table A.2 shows the command line option that specifies the name of the output machine language data file.

Table A.2 Options for Specifying Output Files

Option	Function
-o*filename*	Specifies the name(s) of the file(s) (absolute module file, map file, etc.) generated by link77. This option can also be used to specify the destination directory. Do not specify the filename extension.
-dir	Specifies the destination directory of the file(s) (machine language data file, map file, etc.) generated by link77.

-o *filename*

Output file specification

Function : Specifies the name(s) of the file(s) (absolute module file, map file, etc.) generated by link77. This option can also be used to specify the destination directory. You must NOT specify the filename extension.

Syntax : nc77Δ-o *filename*Δ<C source file>

Execution example :

```
% nc77 -o./test/sample ncrt0.a77 sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

ncrt0.a77
sample.c
% cd test
% ls
total 65
drwxr-x---  2 toolusr       512 Aug 17 16:13 ./
drwxrwxrwx 11 toolusr      3584 Aug 17 16:14 ../
-rw-r-----  1 toolusr     44040 Aug 17 16:14 sample.hex

%
```

* In this example, the option is used to specify that sample.hex, are output to directory ./test.

-dir *directory Name*

Output file specification

Function : This option allows you to specify an output destination for the output file.

Syntax : nc77Δ-dir directory name

Execution example :

```
% nc77 -dir./test/sample -o ncrt0.a77 sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

ncrt0.a77
sample.c
% cd test/sample
% ls
total 65
drwxr-x---  2 toolusr       512 Aug 17 16:13 ./
drwxrwxrwx 11 toolusr      3584 Aug 17 16:14 ../
-rw-r-----  1 toolusr     44040 Aug 17 16:14 ncrt0.a77

%
```

* In this example, the option is used to specify that ncrt0.a77, are output to directory ./test/sample.

A.2.3 Version Information Display Option

Table 2.3 shows the command line options that display the cross-tool version data.

Table 2.3 Options for Displaying Version Data

Option	Function
-v	Displays the name of the command program and the command line during execution
-V	Displays the startup messages of the compiler programs, then finishes processing (without compiling)

-v

Display command program name

Function : Compiles the files while displaying the name of the command program that is being executed

Execution example :

```
% nc77 -c -v sample.c
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

sample.c
cpp77 sample.c -o sample.i -DMELPS -DMELPS7700 -DNC77
ccom77 sample.i -o ./sample.a77
rasm77 -. -N sample.a77

%
```

Notes : Use lowercase v for this option.

-V

Display version data

Function : Displays version data for the command programs executed by the compiler, then finishes processing

Execution example :

```
%nc77 -V
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999, MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

NC77 Compile Driver for 7700 Family Version 4.XX.XX
NC Preprocessor                 Version 4.XX.XX
NC77 Compiler for 7700 Family   Version 2.XX.XX (NC_CORE Version 2.XX.XX)
Relocatable Macro Assembler for 7700 Family Version V.5.XX.XX
NC77 Branch Optimizer for 7700 Family Version 1.XX.XX
7700 Family LINKER V.2.XX.XX
NC77 IEEE-695 Object Format Converter for 7700 Family Version 1.XX.XX
%
```

Supplement : Use this option to check that the compiler has been installed correctly. The Release Notes list the correct version numbers of the commands executed internally by the compiler.
If the version numbers in the Release Notes do not match those displayed using this option, the package may not have been installed correctly. See the "NC77WA V.5.00 Guide" for details of how to install the NC77 package.

Notes :
1. Use uppercase V for this option.
2. If you specify this option, all other options are ignored.

A.2.4 Options for Debugging

Table A.4 shows the command line options for outputting the symbol file for the C source file.

Table A.4 Options for Debugging

Option	Short form	Function
-gie	None.	Outputs an IEEE-695 absolute format file (extension .ie). When debugging your program at the C language level, always specify this option.
-gie_no_local_symbol	-gINLS	Outputs a file in absolute IEEE-695 format (having the extension .ie), but doesn't output local symbols contained in the assembly language file to the IEEE-695 file
-genter	None.	generates a stack flame at calling a function
-g	None.	Outputs the symbol file (extension .sym) required for debugging

-gie

Create IEEE-695 absolute format file

Function: Outputs an IEEE-695 absolute format file (extension .ie)

Notes: When debugging your program at the C language level, always specify this option.

When IEEE-695 absolute format files are read by third-party emulators or simulators, etc., there is a risk that, because of differences such as the existence of data not stipulated by IEEE-695, some functions do not operate correctly or cannot be read. Please note that Mitsubishi Electric Semiconductor Systems Corp. may not be able to resolve such problems. Please see the Release Notes supplied with the NC77 package for details of the operating environment.

-gie_no_local_symbol   -gINLS

Create IEEE-695 absolute format file

Function: Outputs a file in absolute IEEE-695 format (having the extension `.ie`), but doesn't output local symbols contained in the assembly language file to the IEEE-695 file.

Notes: Specifying the option -gie_no_local_symbol generates an IEEE-695 file in the similar format to NC77 V.3.20.

-genter

Generates a stack flame

Function: generates a stack flame at calling a function

Notes: Be sure to specify this option when using the debugger's stack trace function. If this option is specified, a stack frame is always generated at entry to the function regardless of whether a stack frame is needed. Consequently, this causes the ROM capacity and the stack capacity used to increase.

-g

Outputting debugging information

Function : Creates the symbol file (extension .sym) required for debugging.

Note : When debugging your program at the C language level, do not use this option(use -gie).

A.2.5 Optimization Options

Table A.5 shows the command line options for optimizing program execution speed and ROM capacity.

Table A.5 Optimization Options

Option	Short form	Function
-O[1-5]	None.	Effect the best possible optimization both in execution speed and in ROM capacity, level by level.
-OR	None.	Maximum optimization of ROM size followed by speed
-OS	None.	Maximum optimization of speed followed by ROM size
-Oconst	-OC	Performs optimization by replacing references to the const-qualified external variables with constants
-Ono_bit	-ONB	Suppresses optimization based on grouping of bit manipulations
-Ono_break_source_debug	-ONBSD	Suppresses optimization that affects source line data
-Ono_float_const_fold	-ONFCF	Suppresses the constant folding processing of floating point numbers
-Ono_stdlib	-ONS	Inhibits inline padding of standard library functions and modification of library functions.
-Osp_adjust	-OSA	Optimizes removal of stack correction code. This allows the necessary ROM capacity to be reduced. However, this may result in an increased amount of stack being used.
-Ostack_frame_align (NC30,NC77,NC79 Only)	-OSFA	Aligns the stack frame on an even boundary.

Effect of optimization options

Effect	-O	-OR	-OS	-OSA	-OSFA
speed	better	worse	better	better	better
ROM size	better	better	worse	bettter	-
stack using	better	-	-	-	worse

better : turn better(or remains the same).
worse : turn worse(or remains the same).
- :remains unchanged.

-O[1-5]

Optimization

Function : Optimizes speed and ROM size to the maximum. This option can be specified with -g or -gie options.

Supplement : Optimization is performed to obtain the maximum effect on both speed and ROM size.   This option can be specified along with the -g or -gie option.
-O3 is assumed if you specify no numeric(no level).

-O1: Makes -O3, -Ono_bit, -Ono_break_source_debug, -Ono_float_const_fold, -Ono_stdlib valid.

-O2: Same as -O1.

-O3: Effect the best possible optimization both in execution speed and in ROM capacity.

-O4: Makes -O3 and -Oconst valid.

-O5: Makes -O4 valid. And, effect the best possible optimization in common subexpressions (if the option -OR is concurrently specified); effects the best possible optimization in transfer and comparison of character strings(if the option -OS is concurrently specified).

- -O5 doesn't output normal code in such an instance as mentioned below. In an instance in with two or more pointers are present within a single function and they point to an identical address

```
example)
    int    a=3, b;
    int    *p = &a;
    main()
    {
        test1();
    }
    test1()
    {
        *p = a * 3;   /* a = *p = 9 */
        a = 10;       /* a = *p = 10 */
        b = *p;       /* a = *p = b = 10 */
        printf("b = %d(expect b = 10)\n", b);
    }
result)
    b = 9(expect = 10)
```

-OR

Optimization

Function : Optimizes ROM size in preference to speed. This option can be specified with -g or -gie options.

Supplement : When this option is used, the source line information may partly be modified in the course of optimization. If you do not want the source line information to be modified, use the -One_break_source_debug (-ONBSD) option to suppress optimization.

-OS

Optimization

Function : Although the ROM size may somewhat increase, optimization is performed to obtain the fastest speed possible. This option can be specified along with the -g or -gie option.

-Oconst -OC

Optimization

Function : Performs optimization by replacing references to the const-qualified external variables with constants

Supplement : Optimization is performed when the following conditions are satisfied simultaneously :

1. Extern variables excluding structures, unions, and arrays;
2. Extern variables declared using the const qualifier;
3. Extern variables initialized in the same C source file.

The following example shows code that can be optimized.

Code example :

```
int const i = 10;

func()
{
    int k = i;   /* i is replaced with 10. */
          :
          :

}
```

-Ono_bit -ONB

Suppression of optimization

Function : Suppresses optimization based on grouping of bit manipulations

Supplement : When you specify -O (or -OR or -OS), optimization is based on grouping manipulations that assign constants to a bit field mapped to the same memory area into one routine.
Because it is not suitable to perform this operation when there is an order to the consecutive bit operations, as in I/O bit fields, use this option to suppress optimization.

Notes : This option is only valid if you specify option -O (or -OR or -OS).

-Ono_break_source_debug -ONBSD

Suppression of optimization

Function : Suppresses optimization that affects source line data

Supplement : Specifying the -OR or -O option performs the following optimization, which may affect source line data. This option (-ONBSD) is used to suppress such optimization.

Notes : This option is valid only when the -OR or -O option is specified.

-Ono_float_const_fold -ONFCF

Suppression of optimization

Function : Suppresses the constant folding processing of floating point numbers

Supplement : By default, NC77 folds constants. Following is an example.

[before optimization]

```
(val/1000e250)*50.0
```

[after optimization]

```
val/20e250
```

In this case, if the application uses the full dynamic range of floating points, the results of calculation differ as the order of calculation is changed. This option suppresses the constant folding in floating-point numbers so that the calculation sequence in the C source file is preserved.

-Ono_stdlib -ONS

Suppression of optimization

Function : Suppresses inline padding of standard library functions, modification of library functions, and similar other optimization processing.

Notes : Specify this option, when make a function which name is same as standard library function.

-Osp_adjust -OSA

Removing stack correction code after calling a function

Function : Performs optimization to remove stack correction code after calling a function.

Notes : The -Osp_adjust option allows you to reduce the ROM capacity. However, it may cause the amount of stacks used to increase.

-Ostack_frame_align -OSFA

Aligns stack frame

Function: Aligns the stack frame on an even boundary.

Supplement: When even-sized auto variables are mapped to odd addresses, memory access requires one more cycle than when they are mapped to even addresses. This option maps even-sized auto variables to even addresses, thereby speeding up memory access.

Notes:

1. The following functions specified in #pragma are not aligned.

   ●#pragma INTHANDLER
   ●#pragma HANDLER
   ●#pragma ALMHANDLER
   ●#pragma CYCHANDLER
   ●#pragma INTERRUPT [*1]

2. Be sure that the stack point is initialized to an even address in the startup program. Also, be sure to compile all programs using this option.

3. All files should be compiled using this option.

*1. Alignment is not performed on interrupt functions because it is not possible to guarantee that the stack point has an even-value when the interrupt occurs. Therefore, if this option is specified in functions called from an interrupt function, processing times may actually increase.

A.2.6 Options for Selecting Branch Instructions

Table A.6 shows the command line options for selecting branch instructions in the assembly language source files created by nc77.

Table A.6 Branch Instruction Selection

Option	Function
-OB1	Generates branch instructions taking only code size into account (default). The branch instructions are converted in the following order: bra→bral→jmpl
-OB2	Generates branch instructions taking speed into account (in same bank). The branch instructions are converted in the following order: bra→jmp
-OB3	Generates branch instructions taking speed into account (outside bank). The branch instructions are converted in the following order: bra→jmpl

*No the bra instruction is output if you specify -OB2 or -OB3 with -OS.

-OB1

Selection of branch instruction

Function: Selects branch instructions taking only code size into account. The branch instructions are selected according to the following rules:
Order of branch instruction selection: BRA→BRAL→JMPL

Notes: If you do not specify a branch instruction selection option, the branch instructions are selected according to exactly the same rules as -OB1.

-OB2

Selection of branch instruction

Function: Selects branch instructions taking speed into account. Note, however, that execution can only jump to an address in the same bank. An error occurs during linking if the jump cannot be executed with the selected jump instruction. The branch instructions are selected according to the following rules:
Order of branch instruction selection: BRA→JMP

Notes: If you specify -OB2 with -OS, only JMP is selected.

-OB3

Function: Selects branch instructions taking speed into account. The selected branch instruction can jump to any address in memory in the 7700 family. The branch instructions are selected according to the following rules:
Order of branch instruction selection: BRA→JMPL

Notes: If you specify -OB3 with -OS, only JMPL is selected.

A.2.7 Generated Code Modification Options

Table 2.7 shows the command line options for controlling nc77-generated assembly code.

Table A.7(1/2)   Generated Code Modification Options

Option	Short form	Description
-fansi	None.	Makes -fnot_reserve_far_and_near, -fnot_reserve_asm, -fnot_reserve_inline, and -fextend_to_int valid.
-fnot_reserve_asm	-fNRA	Exclude asm from reserved words. (Only _asm is valid.)
-fnot_reserve_far_and_near	-fNRFAN	Exclude far and near from reserved words. (Only _far and _near are valid.)
-fnot_reserve_inline	-fNRI	Exclude far and near from reserved words. (Only _inline is made a reserved word.)
-fextend_to_int	-fETI	Performs operation after extending char-type data to the int type. (Extended according to ANSI standards.)*1
-fchar_enumerator	-fCE	Handles the enumerator type as an unsigned char type, not as an int type.
-fno_even	-fNE	Allocate all data to the odd section , with no separating odd data from even data when outputting .
-fshow_stack_usage	-fSSU	Outputs the usage condition of the stack pointer to a file (extension .stk).
-ffar_RAM_data	-fFRAM	Changes the default attribute of RAM data to far.
-ffar_ROM_data	-fFROM	Changes the default attribute of ROM data to far.
-fall_far	-fAF	Changes all defaults to far types.
-fnear_function	-fNF	Sets the function default to near. Near functions are called with jsr and returned with rts.
-ffar_program_section	-fFPS	Maps near functions and far functions to the program_F section.
-fnot_use_MVN	-fNUM	Suppresses transfer of blocks with the MVN instruction (The MVN instruction is used for assignment among structures.)
-bank=	None.	Specifies the value of the data bank register (DT) at compiling. The default when not specified is 0.
-fswitch_table	-fST	Uses the jump table only when the code size of case statements in switch statements is satisfactory.
-fconst_not_ROM	-fCNR	Does not handle the types specified by const as ROM data.
-fnot_address_volatile	-fNAV	Does not regard the variables specified by #pragma ADDRESS (#pragma EQU) as those specified by volatile.

*1. char-type data or signed char-type data evaluated under ANSI rules is always extended to int-type data. This is because operations on char types (c1=c2*2/c3; for example) would otherwise result in an overflow and failure to obtain the intended result.

Table A.7(2/2) Generated Code Modification Options

Option	Short form	Description
-fsmall_array	-fSA	When referencing a far-type array, this option calculates subscripts in 16 bits if the total size of the array is within 64K bytes.
-fenable_register	-fER	Make register storage class available
-fuse_DIV	-fUD	This option changes generated code for divide operation.

-fansi

Modify generated code

Function : Validates the following command line options:

- ●fnot_reserve_asm .... Removes asm from reserved words
- ●fnot_reserve_far_and_near .. Removes far and near from reserved words
- ●fnot_reserve_inline .... Removes inline from reserved words
- ●fextend_to_int .... Extends char-type data to int-type data to perform operations

Supplement : When this option is specified, the compiler generates code in conformity with ANSI standards.

-fnot_reserve_asm -fNRA

Modify generated code

Function : Removes asm from the list of reserved words. However, _asm, which has the same function, remains as a reserved word.

-fnot_reserve_far_and_near -fNRFAN

Modify generated code

Function : Removes far and near from list of reserved words. However, _far and _near, which have the same functions, remain reserved words.

-fnot_reserve_inline -fNRI

Modify generated code

Function : Does not handle inline as a reserved word. However, _inline that has the same function is handled as a reserved word.

-fextend_to_int -fETI

Modify generated code

Function : Extends char-type or signed char-type data to int-type data to perform operation (extension as per ANSI rules)

Supplement : In ANSI standards, the char-type or singed char-type data is always extended into the int type when evaluated. This extension is provided to prevent a problem in char-type arithmetic operations, e.g., c1 = c2 * 2 / c3; that thechar type overflows in the middle of operation, and that the result takes on an unexpected value. An example is shown below.

```
main()
{
        char   c1;
        char   c2 = 200;
        char   c3 = 2;

        c1 = c2 * 2 / c3;
}
```

In this case, thechar type overflows when calculating [c2 * 2], so that the correct result may not be obtained.
Specification of this option helps to obtain the correct result. The reason why extension into the int type is disabled by default is because it is conducive to increasing the ROM efficiency any further.

-fchar_enumerator -fCE

Modify generated code

Function : Processes enumerator types not as int types but as unsigned char types.

Notes : The type debug information does not include information on type sizes. Therefore, if this option is specified, the enum type may not be referenced correctly in some debugger.

-fno_even -fNE

Modify generated code

Function : When outputting data, does not separate odd and even data. That is, all data is mapped to the odd sections (data_NO, data_FO, data_INO, data_IFO, bss_NO, bss_FO, rom_NO, rom_FO)

Supplement : By default, the odd-size and the even-size data are output to separate sections. Take a look at the example below.

```
char   c;
int   i;
```

In this case, variable "c" and variable "i" are output to separate sections. This is because the even-size variable "i" is located at an even address. This allows for fast access when accessing in 16-bit bus width.
Use this option only when you are using the 7700 family in 8-bit bus width and when you want to reduce the number of sections.

Notes : When #pragma SECTION is used to change the name of a section, data is mapped to the newly named section.

-fshow_stack_usage -fSSU

Modify generated code

Function : Outputs the stack utilization to a file (extension .stk)

Supplement : The Stack Size Calculation Utility stk77 uses the files generated by this option as it calculates the stack sizes used in the program.

Notes : The usage status of the stacks used in the asm function is not output. Nor does the compiler calculate the stacks used in the asm function even when using stk77.

-ffar_RAM_DATA -fFRAM

Modify generated code

Function : Change the default attribute of RAM data to far.

Supplement : Always absolute long addressing mode(32bits width) is used for the RAM data (variables).

-ffar_ROM_DATA -fFROM

Modify generated code

Function : Change the default attribute of RAM data to far.

Supplement : Always absolute long addressing mode(32bits width) is used for the ROM data.

-fall_far -fAF

Modify generated code

Function: Validates all command line options shown below and changes the handling of data, auto pointer address variables, auto variable addresses, and character string data to far. Also sets the default for all functions to far.

- -ffar_RAM_data (-fFRAM)
- -ffar_ROM_data (-fFROM)

-fnear_function -fNF

Modify generated code

Function : Sets the function default to near. Near functions are called with JSR and returned with RTS.

Notes : The runtime library, etc., are set to always be output to the program_F section. Therefore, even when this option is specified, the program_F section is included when the library is linked.
When this option is used to create a program that operates only in bank 0, the program_F section must be mapped to bank 0 as the library section.

-ffar_program_section -fFPS

Modify generated code

Function: Maps near functions and far functions to the program_F section.

Notes: When #pragma SECTION is used to change the name of a section, data is mapped to the newly named section.

-fnot_use_MVN -fNUM

Modify generated code

Function: Suppresses transfer of blocks with the MVN instruction (The MVN instruction is used for assignment among structures.)

Notes: The 7700 family receives no interrupts during execution of MVN. Therefore, when assignments are made among large structures, the ability to respond to interrupts worsens. Use this option in such cases. Note that NC77 does not generate MVP instructions.

-bank=*bank No.*

Function: Specifies the value of the near area bank (DT) at compiling. The default bank No. is 0. The bank No. can be specified in decimal or hexadecimal. When specified in hexadecimal, add 0X or 0x to the front of the number.

Format: nc77D-bank=*bank No.*D<C source file name>

Notes:

1. When you use this option to specify a bank other than No.0, you must also define _DT in the ncrt0.a77 startup program to have the same No.
2. Do not insert any spaces between -bank and the equal sign or between the equal sign and the bank No.

-fswtich_table -fST

Modify generated code

Function: Uses the jump table only when the code size of case statements in switch statements is satisfactory.

Notes: The jump table is not necessarily used even when this option is specified. Also, generated code will not run correctly if the generated jump table is mapped across banks.

-fconst_not_ROM -fCNR

Modify generated code

Function : Does not handle the types specified by const as ROM data.

Supplement : The const-specified data by default is located in the ROM area. Take a look at the example below.

```
int const array[10] = { 1,2,3,4,5,6,7,8,9,10 };
```

In this case, the array "array" is located as ROM. By specifying this option, you can locate the "array" in the RAM area.
You do not normally need to use this option, however

-fnot_address_volatile -fNAV

Modify generated code

Function : Does not handle the global variables specified by #pragma ADDRESS or #pragma EQU or the static variables declared outside a function as those that are specified by volatile.

Supplement : If I/O variables are optimized in the same way as for variables in RAM, the compiler may not operate as expected. This can be avoided by specifying volatile for the I/O variables.
Normally #pragma ADDRESS or #pragma EQU operates on I/O variables, so that even though volatile may not actually be specified, the compiler processes them assuming volatile is specified. This option suppresses such processing.
You do not normally need to use this option, however.

-fsmall_array -fSA

Modify generated code

Function : When referencing a far-type array whose total size is unknown when compiling, this option calculates subscripts in 16 bits assuming that the array's total size is within 64 Kbytes.

Supplement : If when referencing array elements in a far-type array, the total size of the array is uncertain, the compiler calculates subscripts in 32 bits in order that arrays of 64 Kbytes or more in size can be handled.
Take a look at the example below.

```
extern   int   array[];
int   i = array[j];
```

In this case, because the total size of the array "array" is not known to the compiler, the subscript "j" is calculated in 32 bits.
When this option is specified, the compiler assumes the total size of the array "array" is 64 Kbytes or less and calculates the subscript "j" in 16 bits. As a result, the processing speed can be increased and code size can be reduced.
Mitsubishi recommends using this option whenever the size of one array does not exceed 64 Kbytes.

-fenable_register -fER

Register storage class

Function : Allocates variables with a specified register storage class to registers

supplement : When optimizing register assignments of auto variables, it may not always be possible to obtain the optimum solution. This option is provided as a means of increasing the efficiency of optimization by instructing register assignments in the program under the above situation.
When this option is specified, the following register-specified variables are forcibly assigned to registers:

1. Integral type variable
2. Pointer variable

Note : Because register specification in some cases has an adverse effect that the efficiency decreases, be sure to verify the generated assembly language before using this specification.

-fuse_DIV -fUD

Changes generated code

Function : This option changes generated code for divide operation.

supplement : For divide operations where the dividend is a 4-byte value, the divisor is a 2-byte value, and the result is a 2-byte value, the compiler generates div and divs (only 775x) microcomputer instructions.

Note : The div instruction of the 7700 has such a characteristic that when the operation resulted in an overflow, the result becomes indeterminate. Therefore, when the program is compiled in default settings by NC77, it calls a runtime library to correct the result for this problem even in cases where the dividend is 4-byte, the divisor is 2-byte, and the result is 2-byte.
If the divide operation results in an overflow when this option is specified, the compiler may operate differently than stipulated in ANSI.

A.2.8 Warning Options

Table A.8 shows the command line options for outputting warning messages for contraventions of nc77 language specifications.

Table A.8 Warning Options

Option	Short form	Function
-Wnon_prototype	-WNP	Outputs warning messages for functions without prototype declarations.
-Wunknown_pragma	-WUP	Outputs warning messages for non-supported #pragma.
-Wno_stop	-WNS	Prevents the compiler stopping when an error occurs.
-Wstdout	None.	Outputs error messages to the host machine's standard output (stdout).
-Werror_file<file name>	-WEF	Outputs error messages to the specified file.
-Wstop_at_warning	-WSAW	Stops the compiling process when a warning occurs.
-Wnesting_comment	-WNC	Outputs a warning for a comment including */ .
-Wccom_max_warnings	-WCMW	This option allows you to specify an upper limit for the number of warnings output by nc77.
-Wall	None.	Displays message for all detectable warnings.
-Wmake_tagfile	-WMT	Outputs error messages to the tag file of source-file by source-file.
-Wuninitialize_variable	-WUV	Outputs a warning about auto variables that have not been initialized.
-Wlarge_to_small	-WLTS	Outputs a warning about the tacit transfer of variables in descending sequence of size.

-Wnon_prototype -WNP

Warning option

Function : Outputs warning messages for functions without prototype declarations or if the prototype declaration is not performed for any function

supplement : Function arguments can be passed via a register by writing a prototype declaration.
Increased speed and reduced code size can be expected by passing arguments via a register. Also, the prototype declaration causes the compiler to check function arguments. Increased program reliability can be expected from this.
Therefore, Mitsubishi recommends using this option whenever possible.

-Wunknown_pragma -WUP

Warning option

Function : Outputs warning messages for non-supported #pragma

supplement : By default, no alarm is generated even when an unsupported, unknown "#pragma" is used.
When you are using only the NC-series compilers, use of this option helps to find misspellings in "#pragma."
When you are using only the NC-series compilers, Mitsubishi recommends that this option be always used when compiling.

-Wno_stop -WNS

Warning option

Function : Prevents the compiler stopping when an error occurs

supplement : The compiler compiles the program one function at a time. If an error occurs when compiling, the compiler by default does not compile the next function. Also, another error may be induced by an error, giving rise to multiple errors. In such a case, the compiler stops compiling.
When this option is specified, the compiler continues compiling as far as possible.

Note : A descriptive error may cause a System Error. In such a case, compilation stops.

-Wstdout

Warning option

Function : Outputs error messages to the host machine's standard output (stdout)

Supplement : Use this option to save error output, etc. to a file by using Redirect in the MS-Windows95 version (personal computer version).

Note : In NC77 for MS-Windows95 version(personal computer version), errors from rasm77 and link77 invoked by the compile-driver are output to the standard output regardless of this option.

-Werror_file <file name> -WEF

Warning option

Function : Outputs error messages to the specified file

Syntax : nc77Δ-Werror_fileΔ<output error message file name>

Supplement : The format in which error messages are output to a file differs from one in which error messages are displayed on the screen. When error messages are output to a file, they are output in the format suitable for the "tag jump function" that some editors have.

Output example:

```
test.c12       Error(ccom):unknown variable i
```

-Wstop_at_warning -WSAW

Warning option

Function : When a warning occurs, the compiler's end code is set to "10" as it is returned.

Supplement : If a warning occurs when compiling, the compilation by default is terminated with the end code "1" (terminated normally).
Use this option when you are using the make utility, etc. and want to stop compile processing when a warning occurs.

-Wnesting_comment -WNC

Warning option

Function : Generates a warning when comments include "/*"

Supplement : By using this option, it is possible to detect nesting of comments.

-Wccom_max_warnings   -WCMW

Warning option

Function : This option allows you to specify an upper limit for the number of warnings output by nc77.

Supplement : By default, there is no upper limit to warning outputs.
Use this option to adjust the screen as it scrolls for many warnings that are output.

Note : For the upper-limit count of warning outputs, specify a number equal to or greater than 0. Specification of this count cannot be omitted. When you specify 0, warning outputs are completely suppressed inhibited.

-Wall

Warning option

Function : Displays message for all detectable warnings, which are displayed with the -Wnon_prototype(-WNP) and -Wunknown_pragma(WUP) options and in the following cases (1) and (2). Note that these warnings are not all coding errors because they are the compiler's inference.

Case (1)
When the assignment operator = is used in the if statement, the for statement or a comparison statement with the && or || operator.

```
Example:   if( i = 0 )
               func();
```

Case (2)
When "==" is written to which '=' should be specified.

```
Example:   i == 0;
```

Case(3)
When function is defined in old format.

```
Example:   func(i)
           int   i;
           {
                  :
             (omitted)
                  :
           }
```

Note : These alarms are detected within the scope that the compiler assumes on its judgment that description is erroneous. Therefore, not all errors can be alarmed.

-Wmake_tagfile -WMT

Warning option

Function : Outputs error messages to the tag file of source-file by source-file, when an error or warning occurs.

Supplement : This option with ì-Werror_file<file name>î(-WEF) option canít specify.

-Wuninitialize_variable -WUV

Warning option

Function : Outputs a warning about auto variables that have not been initialized.

-Wlarge_to_small -WLTS

Warning option

Function : Outputs a warning about the tacit transfer of variables in descending sequence of size.

A.2.9 Assemble and Link Options

Table A.9 shows the command line options for specifying rasm77 and link77 options.

Table A.9 Assemble and Link Options

Option	Function
-rasm77Δ<*option*>	Specifies options for the rasm77 link command. If you specify two or more options, enclose them in double quotes.
-link77Δ<*option*>	Specifies options for the link77 assemble command. If you specify two or more options, enclose them in double quotes.

-rasm77 *"option"*

Assemble/link option

Function : Specifies rasm77 assemble command options
If you specify two or more options, enclose them in double quotes.

Syntax : nc77∆-rasm77∆*"option1∆option2"*∆<C source file>

Execution example : In the example below, the assembler list file is generated when compiling.

```
% nc77 -c -v -rasm77 "-l -s" sample.c
NC77 COMPILER for 7700 FAMILY V.5.00 Release 1
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

sample.c
cpp77 sample.c -o sample.i -DMELPS -DMELPS7700 -DNC77
ccom77 sample.i -o ./sample.a77
loop77 -zopt77 -. -l -s sample.a77
% ls sample.*
-rw-r-----  1 toolusr      2059 Aug 17 15:43 sample.a77
-rw-r--r--  1 toolusr      2850 Aug 17 14:51 sample.c
-rw-r-----  1 toolusr       597 Aug 17 15:43 sample.ext
-rw-r-----  1 toolusr     10508 Aug 17 15:43 sample.prn     ←
-rw-r-----  1 toolusr       587 Aug 17 15:43 sample.r77
%
```

* In this example, generation of a print file (extension .prn) is specified as an option of the assemble command.

Note :
1. When rasm77 is started by the branch optimizer loop77, the rasm77 startup options -. and -C are specified automatically.
2. Do not specify the RASM77 options -B, -E, -O or -Q.

For reference, the following table lists the RASM77 V.5.00 options.

Option	Function
-.	Suppresses the output of all messages to the screen. Use this option when running RASM77 from a batch file to prevent anything being displayed on the screen.
-B	Checks the bit size conformity. When you specify this option, warning 6 is output when referencing local labels declared in pseudo instructions".BYTE",".WORD",".BLKB" or ".BLKW" if they do not match the bit size declared in ".DATA" or ".INDEX". * Do not specify this option.
-C	Outputs source debug information to an object file. Specify this option during assembling when debugging source code.
-D	Sets numerical values in symbols. Symbols set by commands are handled in the same way as symbols defined by pseudo instruction .EQU.
-E	Generates a tag file (extension .tag) and starts the editor
-L	Generates a print file (extension .prn). If not specified, no print file is generated. (However, if option -M is specified, a print file is generated even if -L is not specified.)
-LC	Outputs to the print file even the parts that do not satisfy the conditions when performing a conditional assemble using .IF. If not specified, the parts are not output to the print file.
-M	Outputs the macro expansion to the print file and generates the print file. If this option is not specified, no macro expansion is output to the print file.
-Q	Outputs a warning when resetting using pseudo instruction ".EQU" . If this option is not specified, no error occurs when different values are set for the same symbol. *Do not specify this option.
-O	Specifies the destination path for the generated file. You can specify a drive and/or directory. If this option is not specified, the files are output to the same directory as the source file.
-S	Outputs local symbol information to an object file
-U	Allows the colon (:) following labels to be omitted
-X	Starts the CRF77 cross-referencer on completion of assembling

* NC77 allows you to use option -rasm77 to control the assembler. However, you cannot specify RASM77 options -B, -E, -O or -Q in this case.

- NC77 generates code so that no warning messages are output during assembling. However, if you specify RASM77 options -B, -E, -O or -Q RASM77 outputs warning messages not used in NC77 and compiling may be aborted.

-link77 *"option"*

Assemble/Link Option

Function : Specifies options for the link77 link command. You can specify a maximum of four options.
If you specify two or more options, enclose them in double quotes.

Syntax : nc77Δ-link77Δ*"option1Δoption2"*Δ<C source file name>

Execution example : In the example below, the map file is generated when compiling.

```
% sun:toolusr(361)-> nc77 -g -v -osample -link77 -ms ncrt0.a77 sample.c
NC77 COMPILER for 7700 FAMILY V.5.00 Release 1
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

ncrt0.a77
loop77 -zopt77 -. -C ncrt0.a77
sample.c
cpp77 sample.c -o sample.i -DMELPS -DMELPS7700 -DNC77
ccom77 sample.i -o ./sample.a77
loop77 -zopt77 -. -C sample.a77


link77 ncrt0.r77 sample.r77 , , , -. -s -M -ms -o. -fsample ,
processing "ncrt0.r77"
processing "sample.r77"
processing "Libraries"
processing "ncrt0.r77"
processing "sample.r77"
processing "/usr3/tool/toolusr/work77/lib77/nc77lib.lib ( fprintf.r77 )"
         :
    (abbreviated)
         :
% ls sample.*
-rw-r--r--  1 toolusr      2850 Aug 17 14:51 sample.c
-rw-r-----  1 toolusr     44040 Aug 17 15:47 sample.hex
-rw-r-----  1 toolusr      8310 Aug 17 15:47 sample.map     ←
-rw-r-----  1 toolusr      8819 Aug 17 15:47 sample.sym
%
```

* In this example, the link command option specifies generation of a map file (extension .map).

Notes : When link77 is started by nc77, link77 startup options -o and -f are automatically applied.
Option -o specifies the output file directory. Option -f specifies the name of the first input file (extension .a77, .r77, or .c) on the nc77 command line. To specify a different directory or filename, specify the output directory and filename in the nc77 command line option -o.

For reference, the following table lists the options for LINK77, which is part of the RASM77 V.5.00 package.

Option	Function
-A	Enables overlapping of absolute-attribute sections with the same name. This is useful in linking when sharing global memory areas.
-C	Outputs a warning if a specific branch instruction is on a bank boundary. However, a warning message is output if data that is the same value as the instruction's machine language exists.
-F	Specifies the output file name
-M	Outputs a map file (extension .map) (section data only)
-MS	Outputs a map file with global label and global symbol lists
-N	Ignores reference data for relocatable files (extension .R77) and library files (extension .LIB) specified in the source file in pseudo instructions .OBJ and .LIB
-O	Specifies the output file directory
-S	Outputs a symbol file (extension .sym)
-V	Checks compatibility of relocatable file versions. However, for compatibility to be checked, the pseudo instruction .VER must be used in the assembly language source files to specify the versions.
-W	Maps sections using word alignment

* NC77 allows you to use option -link77 to control the linker. However, you cannot specify LINK77 options -F or -O in this case.

A.2.10 7750/7751-Compatible Code Generation Option

Table 2.11 shows the command line option for specifying that NC77 generates 7750/7751-compatible code.

Table 2.11 7750/7751-Compatible Code Generation Option

Option	Shortform	Function
-m7750	None.	Generates code that is compatible with the 7750/7751 series

-m7750

Function: Generates code that is compatible with the 7750/7751 Series

Notes: If you specify this option, you must use the 7750/7751 Series-compatible library file.

A.2.11 Miscellaneous Option

Table A.11 shows the command line option for processing the assembly language source files generated by nc77.

Table A.11 Miscellaneous Option

Option	Short form	Function
-dsource	-dS	Outputs C source code as comments in the output assembly language source list

-dsource -dS

Comment option

Function : Outputs the C source code as comments in the output assembly language source list . Validates, when specifies with option -S.

Supplement : When the -S option is used, the -dsouce option is automatically enabled.
Use this option when you want to output C-language source lists to the assembly list file.

A.3 Notes on nc77 Command Line Options

A.3.1 Coding nc77 Command Line Options

The NC77 command line options differ according to whether they are written in uppercase or lowercase letters. Some options will not work if they are specified in the wrong case.

A.3.2 Priority of Options for Controlling nc77

If you specify both the following options in the NC77 command line, the -S option takes precedence and only the assembly language source files will be generated.

- -c : Stop after creating relocatable files.
- -S : Stop after creating assembly language source files.

Appendix B
Extended Functions Reference

To facilitate its use in systems using the 7700 series, NC77 has a number of additional (extended) functions.

This appendix B describes how to use these extended functions, excluding those related to language specifications, which are only described in outline.

Table B.1 Extended Functions (1/2)

Extended function	Content of function
near and far modifiers	1. Specifies whether to use absolute or absolute long addressing mode for data access near ..... Access within same bank (64KB area) far .... Access outside bank (area over 64KB) 2. Specifies whether to use JSR or JSRL instruction for calling functions near ..... Call function using JSR far .... Call function using JSRL
asm function	1. Allows assembly language to be directly incorporated in C programs. Assembly language can even be used outside functions. Example : `asm("LDA A,DP:1");` 2. Allows the compiler to switch the m and x flags in the processor status register Example : `asm(0,0); /* CLP m,x */` 3. Allows the DP offset of storage class AUTO variables to be specified using the name of the variable Example 1 : `asm("LDA  A,DP:$$",i);` Example 2 : `asm("LDA  A,DP:$$",s.i);` Example 3 : `asm("LDA  A,DP:$$",a[3]);` 4. Allows dummy asm functions to be used to selectively suppress optimization Example : asm();
Japanese characters	1. Permits you to use Japanese characters in character strings. Example : `L"漢字"` 2. Permits you to use Japanese characters for character constants. Example : `L'漢'` 3. Permits you to write Japanese characters in comments. Example : `/* 漢字*/` * Shift-JIS and EUC code are supported ,but can't use the half size character of Japanese-KATA-KANA.

Table B.2   Extended Functions (2/2)

Extended feature	Description
Default argument declaration for function	1.Default value can be defined for the argument of a function. Example 1 : `extern int func(int i=1, char c=0);` Example 2 : `extern int func(int i=a, char c=0);` * When writing a variable as a default value, be sure to declare the variable used as a default value before declaring the function. * Write default values sequentially beginning immediately after the argument.
Inline storage class	1. Functions can be inline developed by using the inline storage class specifier. Example : `inline func( int i );` * Always be sure to define the body of an inline function before using the inline function.
Extension of Comments	1. You can include C++-like comments ("//"). Example : `// This is a comment.`
#pragma Extended functions	You can use extended functions for which the hardware of 7700 family in C language.
macro assebler function	You can describe some assembler command as the function of C language. Exampe : char dadd_b(char val1, char val2); Example : char dadd_w(int val1, int val2);

B.1 Near and far Modifiers

In the 7700 family, the addressing mode for referencing and mapping data and calling functions on each side of the bank (64KB) boundary. The addressing mode is controlled in NC77/NC79 using the near and far modifiers.

This chapter describes the specifications of the near and far modifiers.

B.1.1 Overview of near and far Modifiers

The addressing modes of the 7700 family can be broadly classified as follows:

1.Direct addressing mode
2.Absolute addressing mode
3.Absolute long addressing mode

The near and far modifiers select the addressing mode used for variables and functions.

lnear modifier .... Absolute addressing mode
(16-bit addresses)
lfar modifier .... Absolute long addressing mode
(32-bit addresses)

In NC77, the direct page register (DPR) is used as a frame pointer. Therefore, the direct addressing mode cannot be controlled using the near and far modifiers.

The near and far modifiers are added to the type specifier when declaring variables and functions. If you do not specify near or far when declaring a variable or function, NC77 assumes the following attributes:

lVariables .... near attribute
lFunctions................................ far attribute

NC77 also allows you to change these default attributes using an nc77 compile driver command line option.

B.1.2   Format of Variable Declaration

The near and far modifiers are included in declarations using the same syntactical format as the const and volatile type modifiers. Figure B.1 is a format of variable declaration.

```
type specifierΔnear or farΔvariable;
```

Figure B.1   Format of Variable added near / far modifier

Figure B.2 is an example of variable declaration. Figure B.3 is a memory map for that variable

```
int near in_data;
int far  if_data;

func()
{
        (remainder omitted)
          :
```

Figure B.2   Example of Variable Declaration

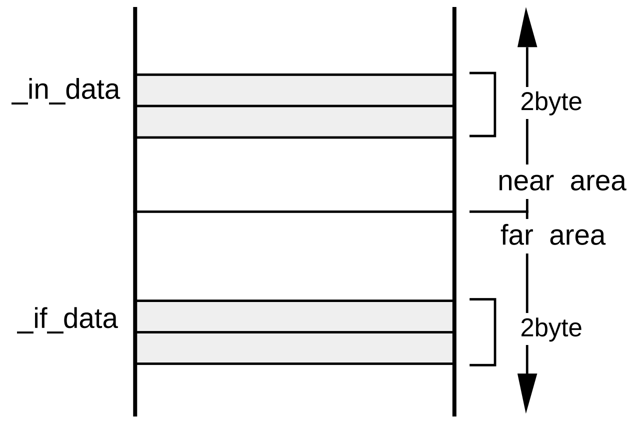


Figure B.3   Memory Location of Variable

B.1.3 Format of Pointer type Variable

Pointer-type variables by default are the near-type (2-byte) variable. A declaration example of pointer-type variables is shown in Figure B.4.

● Example

```
int  *  ptr;
```

Figure B.4 Example of Declaring a Pointer Type Variable(1/2)

Because the variables are located near and take on the variable type near, the description in Figure B.4 is interpreted as in Figure B.5.

● Example

```
int  near  *  near  ptr;
```

Figure B.5 Example of Declaring a Pointer Type Variable(2/2)

The variable ptr is a 2-byte variable that indicates the int-type variable located in the near area. The ptr itself is located in the near area.

Memory mapping for the above example is shown in Figure B.6.

Figure B.6 shows memory maps for above examples.

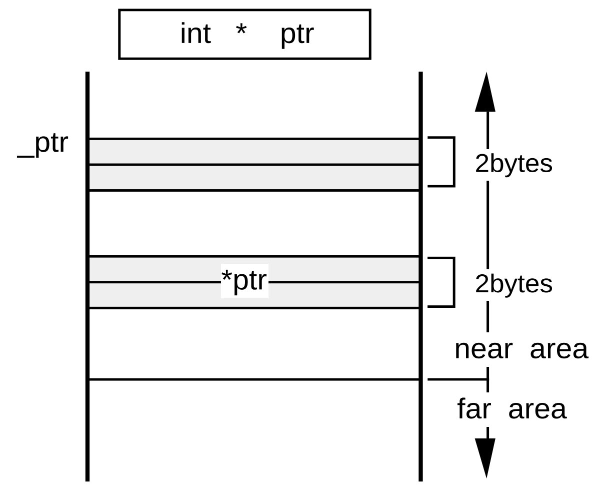


Figure B.6 Memory Location of Pointer type Variable

When near/far is explicitly specified, determine the size of the address at which to store the variable/function that is written on the right side. A declaration of pointer-type variables that handle addresses is shown in Figure B.7.

● Example 1
```
int far *ptr1;
```

● Example 2
```
int * far ptr2;
```

Figure B.7 Example of Declaring a Pointer Type Variable(1/2)

As explained earlier, unless near/far is specified, the compiler handles the variable location as "near" and the variable type as "near." Therefore, Examples 1 and 2 respectively are interpreted as shown in Figure B.8.

● Example 1
```
int far * near ptr1;
```

● Example 2
```
int near * far ptr2;
```

Figure B.8 Example of Declaring a Pointer Type Variable(2/2)

In Example 1, the variable ptr1 is a 4-byte variable that indicates the int-type variable located in the far area. The variable itself is located in the near area. In Example 2, the variable ptr2 is a 2-byte variable that indicates the int-type variable located in the near area. The variable itself is located in the far area.

Memory mappings for Examples 1 and 2 are shown in Figure B.9.

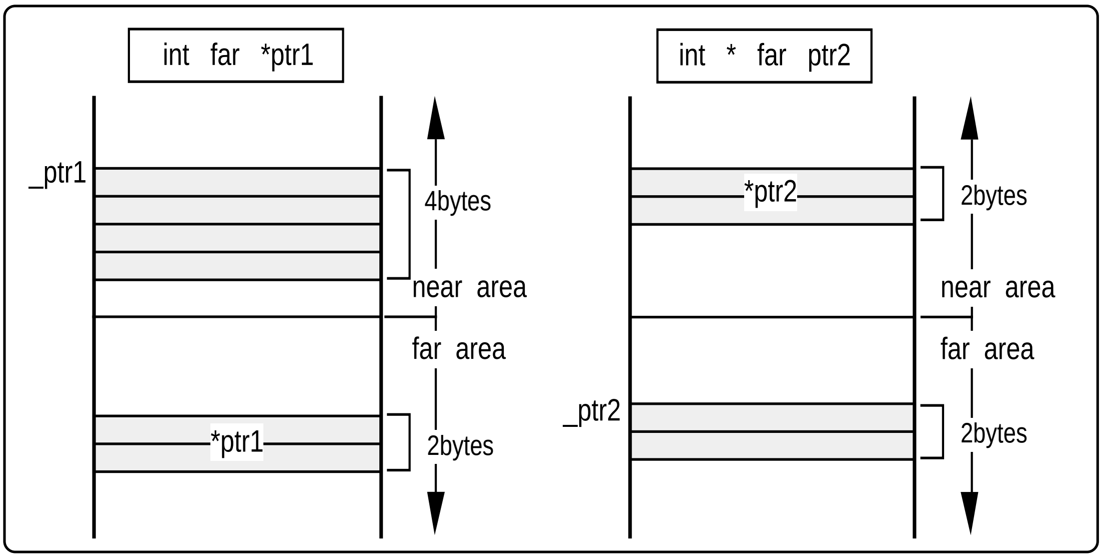


Figure B.9 Memory Location of Pointer type Variable

B.1.4 Format of Function Declaration

Figure B.10 shows the format for the declaration. Figure B.11 is an example declaration.

```
type specifierΔnear or farΔfunction;
```

Figure B.10 Format of Function added near / far modifier.

```
void near func1( void );
int far func2( int );

void near func1()
{
        :
    (abbreviated)
        :
    func2( idata );
}

int far func2( x )
int x;
{
        :
    (abbreviated)
        :
    return x;
}
```

Figure B.11 Example of Variable / Function Declaration

In the example shown in Figure B.11, function func2 is declared as being mapped to a bank other than the near area.

The call and return instructions differ according to whether a function is declared with the near or far modifier.

lFunction with near attribute (calling) ..... JSR [return] .... RTS
lFunction with far attribute (calling) ..... JSRL [return] .... RTL

Figure B.13 shows a memory map for the example in Figure B.11 and shows the relationship between the function call and return.

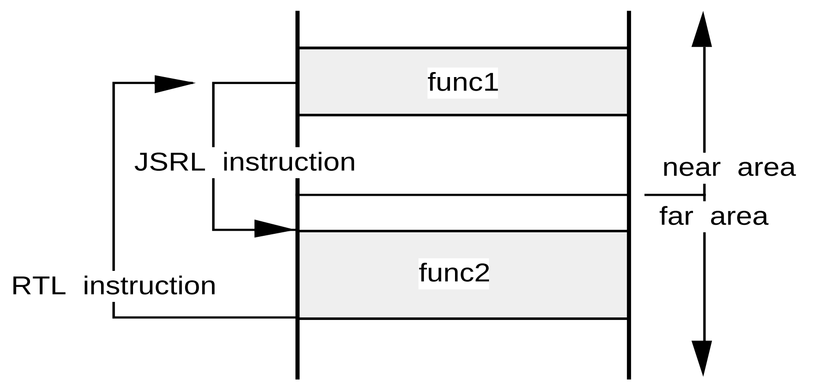


Figure B.13 Relationship of Function Call and Return and Memory Location of Function

B.1.5 near / far Control by nc77 Command Line Options

NC77 handles functions as belonging to the far attribute and variables (data) as belonging to the near attribute if you do not specify the near and far attributes. NC77's command line options allow you to modify the default attributes of functions and variables (data). These are listed in the table below.

Table B.3 nc77 Command Line Options

Option	Compacted form	Description
-fansi	None	Makes -fnot_reserve_far_and_near, -fnot_reserve_inline -fnot_reserve_asm, and -fextend_to_int valid.
-fnot_reserve_far_and_near	-fNRFAN	Exclude far and near from reserved words. (Only _far and _near are valid.)
-fall_far	-fAF	Changes all defaults to the far type.
-fnear_function	-fNF	Changes defaults for functions to near. The near function is called by jsr and returned by rts.
-ffar_program_section	-fFPS	Allocate the near function and the far function to the program_F section.
-ffar_ROM_data	-fFROM	Assumes far as the default attribute of ROM data.
-ffar_RAM_data	-fFRAM	Assumes far as the default attribute of RAM data.

B.1.6 Function of Type conversion from near to far

The program in Figure B.14 performs a type conversion from near to far.

```
int func( int far * );
int far *f_ptr;
int near *n_ptr;

main()
{
    f_ptr = n_ptr;    /* assigns the near pointer to the far pointer */
         :
      (abbreviated)
         :
    func ( n_ptr );   /* prototype declaration for function with far pointer to parameter */
                      /* specifies near pointer parameter at the function call */
}
```

Figure B.14 Type conversion from near to far

When converting type into far, 0 (zero) is stored into high-order address.

B.1.7 Checking Function for Assigning far Pointer to near Pointer

When compiling, the warning message "assign far pointer to near pointer, bank value ignored" is output for the code shown in Figure B.15 to show that the high part of the address (the bank value) has been lost.

```
int func( int near * );
int far *f_ptr;
int near *n_ptr;

main()
{
    n_ptr = f_ptr;      /* Assigns a far pointer to a near pointer */
        :
      (abbreviated)
        :
    func ( f_pyr );     /* prototype declaration of function with near pointer in parameter */
                       /* far pointer implicitly cast as near type */

    n_ptr = (near *)f_ptr;    /* far pointer explicitly cast as near type */
}
```

Figure B.15 Type conversion from far to near

The warning message "far pointer (implicitly) casted by near pointer" is also output when a far pointer is explicitly or implicitly cast as a near pointer, then assigned to a near pointer.

B.1.8 Function for Specifying near and far in Multiple Declarations

As shown in Figure B.16, if there are multiple declarations of the same variable, the type information for the variable is interpreted as indicating a combined type.

```
extern int far idata;
int idata;
int idata = 10;

func()
{
    (remainder omitted)
        :
```

This declaration is interpreted as the following:

```
extern int far idata = 10;

func()
{
    (remainder omitted)
        :
```

Figure B.16   Integrated Function of Function Declaration

As shown in this example, if there are many declarations, the type can be declared by specifying near or far in one of those declarations. However, an error occurs if there is any contention between near and far specifications in two or more of those declarations.*

You can ensure consistency among source files by declaring near or far using a common header file.

common header file
common.h

```
extern int far          func()
```

C source file
a.c

```
#include   "common.h"
main()
{
      func();
}
```

C source file
b.c

```
#include   "common.h"
func()
{
      :
}
```

Figure B.17   Example of Common header file Declaration

* Most near and far mismatches with variables can be found when linking. However, the consistency of functions cannot be found either when compiling or linking. If there is an inconsistency between the calling and called functions, the program will not run properly.

B.1.9 Near and far Attributes of Functions

a. Notes on near and far Attributes of Functions

As shown in Figure 3.18, the JSR instruction is used to call a near function that is mapped to the same program bank as the function, and the RTS instruction is used to return from the function. Similarly, the JSRL instruction is used to call a function regardless if it is in or out of the program bank to which the function is mapped, and RTL is used to return from the function.

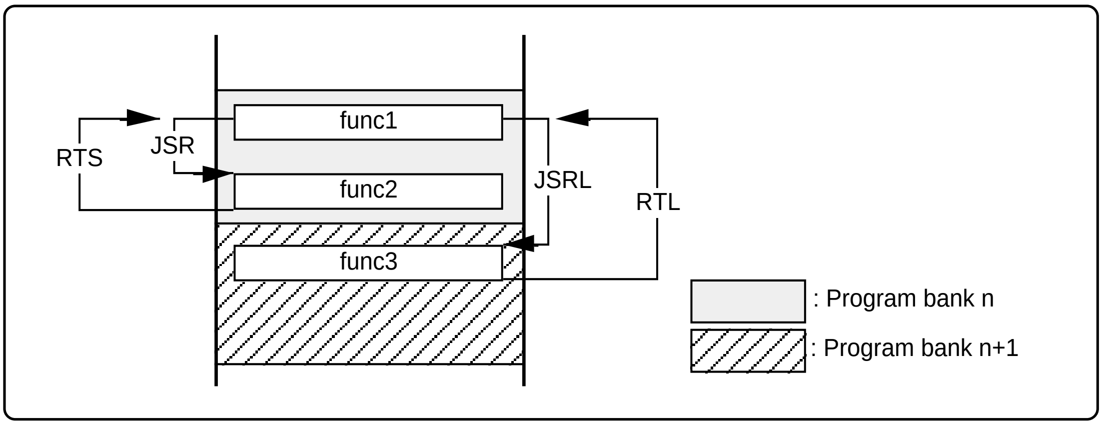


Figure B.18 Relationship of Function Call and Return Instructions and Program Bank

Thus, the instruction used to call a function depends on whether it is inside or outside the program bank, and any mismatch between the calling and called functions will result in the program not running properly. Also, there are no checks for such mismatches during compiling and linking.

The call instruction and return instruction are determined by whether the near or far attribute is specified for the function in its prototype declaration (if there is no prototype declaration, the function takes the far attribute by default). The following examples are based on Figure B.18. Program bank n is assumed to be in the near area.

l Calling func2 from func1
  For func1, func2 has the near attribute. Specify near in the prototype declaration for func2.
l Calling func3 from func1
  For func1, func3 has the far attribute. Specify far in the prototype declaration for func3.
l Calling func1 from func3
  For func1, func3 has the far attribute. If your program includes any such calls, specify far in the prototype declaration for func1.

When the prototype declarations for func1, func2 and func3 are written in a common header file, use the format shown in Figure 16.9. Doing so ensures conformity between calling and called functions.

```
extern int far      func1(void);
extern int near     func2(void);
extern int far      func3(void);
```

Figure B.19   Example of Prototype Declaration

b. Handling Function Addresses

If you do not specify the near or far attribute for functions and variables, the functions are treated as having the far attribute and variables as having the near attribute. The number of bits in the address of functions with no near or far attribute differs from that of normal variables:

- Function address................ 32 bits
- Variable address .... 16 bits

You cannot therefore assign a function address to a char* type or void* type variable (an error results). To handle the variable's address as a 32-bit address, it must be declared as a char far * type or void far * type.

```
func()
{
        int       (* func_ptr)();
        char far  * ptr;

        func_ptr = func;
        ptr = func_ptr;
}
```

Figure B.20   the Relations between the Value of Function Address and the Value of Variable

B.1.10   Notes on near and far Attributes

a. Notes on near and far Modifier Syntax

Syntactically, the near and far modifiers are identical to the const modifier.The following code therefore results in an error.

```
int     i, far          j;      ⇐This is not permitted.

            ▼

int     i;
int     far     j;
```

Figure B.21   Example of Variable Declaration

B.1.11 Notes on near and far Attributes

If neither the near nor far attribute is specified in NC77, functions take the far attribute and variables (data) take the near attribute by default. However, NC77 provides an option that allows you to change these defaults. Figure B.12 shows the relationship between the function or variable (data) size and the option specifying the default attributes.

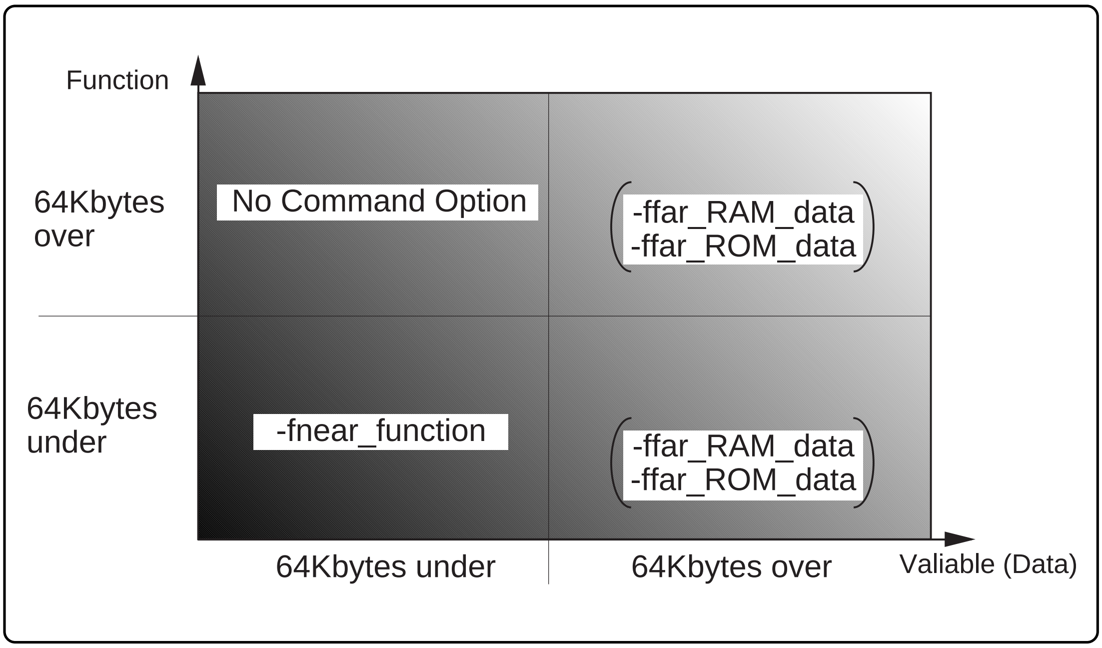


Figure B.22 The Relations between Size of the Functions and Variables and Command line options

The following is the contents of command line options shown above.

Table B.4 Command Line Options of nc77

Command Line Option	Feature
-fnear_function(-fNF)	Changes defaults for functions to near. The near function is called by jsr and returned by rts.
-ffar_ROM_data(-fFROM)	Changes defaults for ROM data to far.
-ffar_RAM_data(-fFRAM)	Changes defaults for RAM dara to far.

B.1.12 Notes on Changing the Bank Value of near Area

Note the following if you specify the nc77/nc79 command line option "-bank=" to specify the bank value of the near area.

1. Do not assign the address of storage class auto variables to a near pointer variable, or as a function parameter.
2. Do not use fgetc, fputc or other I/O function.
3. Specify the same bank value in the "-bank=" option for all source files when they are compiled.
4. When initializing the data bank register in the ncrt0.a77 startup program, set the same bank value in _DT as specified in "-bank=". Also initialize the processor mode register as shown in Figure B.23.

```
start:
;---------------------------------------------------------------------
; after reset,this program will start
;---------------------------------------------------------------------
__DT    .equ        02h
        ldt     #__DT                   ; Initialize data bank register
        sem
        lda.W   A,#24H                   ; set processor mode register
        sta.W   A,LG:5eH
```

* In this example, the bank value of the near area is set to 2 in the NC77 command line option "-bank=2".

Figure B.23 Example of Setting Data Bank Register and Processor Mode register(ncrt0.a77)

B.1.13 Notes on far Bitfield Structures

An error may occur during linking as a result of the coding of bitfield structures that satisfy all the following conditions:

- bitfield structures with the far attribute
- mapped across two or more banks
- assigning constants to members that are referenced with the period operator

A link error (Expression is out of DT range) occurs when a constant is assigned to a member, of a bitfield structure that satisfies all the above conditions, mapped to a different bank from that to which the beginning of a structure is mapped.

In this case, either map the bitfield structure itself to an area that does not span two or more banks, or order the members to which the constants are to be assigned so that they are inside the bank to which the beginning of the structure is mapped.

B.2 asm Function

NC77 allows you to include assembly language routines (asm functions) in your C source programs. The asm function also has extended functions for manipulating the m and x flags and referencing auto variables written in C.

B.2.1 Overview of asm Function

The asm function is used for including assembly language code in a C source program. As shown in Figure B.24, the format of the asm function is asm(" ");, where an assembly language instruction that conforms to the RASM77 language specifications is included between the double quote marks.

```
#pragma ADDRESS ta0_int   55H
char           ta0_int;

void func()
{
         :
      (abbreviated)
         :
        ta0_int = 0x07;                    ⇐Permits timer A0 interrupt
        asm("          CLI");              ⇐Clears interrupt disable flag
}
```

Figure B.24   Example of Description of asm Function (1/2)

Compiler optimization based on the positional relationship of the statements can be partially suppressed using the code shown in Figure B.25.

```
        asm();
```

Figure B.25   Example of Coding asm Function(2/2)

The asm function used in NC77 not only allows you to include assembly language code but also has the following extended functions:

- Specifying the DP offset of storage class auto variables in the C program using the names of the variables in C
- Specifying the data size selection flag (m) and index register size selection flag (x) using the format asm(MFLAG, XFLAG)
- Specifying the register name of storage class register variables in the C program using the names of the variables in C
- Specifying the symbol name of storage class extern and static variables in the C program using the names of the variables in C

* Do not use the format asm(" sem");, etc., to manipulate the m and x flags. Use the format asm(MFLAG, XFLAG). If you include bra or other jump instruction in the asm function, no jump information is passed to the compiler and you should therefore check the generated code after the jump.

B.2.2 Function of Switching the m and x flag

You can use the format in Figure B.26 to switch the data size selection flag (m) and index register size selection flag (x) in the processor status register.

```
asm(MFLAG, XFLAG);

MFLAG:Status of data size selection flag
XFLAG:Status of index register size selection flag
status:0 ............. Flag cleared
       1 ............. Flag set
       2 ............. Do not change flag status
```

Figure B.26 Format for Changing m and x Flags

When changing the status of the m or x flag, use the format shown in Figure B.26. Also, although asm functions can be written outside functions in the C source program, but the m and x flag status must be changed inside functions. Figure B.27 shows examples of switching the m and x flags and the result of compiling.

●C source file

```
void near func()
{
        asm( 0, 0 );            ←Clear m,x
            :
          (abbreviated)
            :
        asm( 1, 1 );            ←Set m,x
            :
          (abbreviated
            :
        asm( 2, 0 );            ←Clear x
            :
          (abbreviated)
            :
        asm( 0, 2 );            ←Clear m

         (remainder omitted)
            :
```

●Assembly language source file (result of compile)

```
;### C_SRC :     asm( 0, 0 );
    clp m,x
      :
    (abbreviated)
      :
;### C_SRC :     asm( 1, 1 );
    sep m,x
      :
    (abbreviated)
      :
;### C_SRC :     asm( 2, 0 );
    clp x
      :
    (abbreviated)
      :
;### C_SRC :     asm( 0, 2 );
    clm
```

Figure B.27 Examples of Switching the m and x flag

B.2.3 Specifying DP Offset Value of auto Variable

The variables of storage class auto (including arguments) written in C language can be referenced and allocated using an offset relative to the direct page register (DP).

By using the description format in Figure B.28, you can use auto variables in the asm function.

```
asm("     op-code   A, DP:$$", auto variable name);
```

Figure B.28 Description Format for Specifying DP Offset

Only one variable name can be specified by using this description format. The following types are supported for variable names:

- Variable name
- Array name [integer]
- Struct name, member name

```
void func()
{
        int idata;
        int a[3];
        struct TAG{
                int  i;
                int  k;
        } s;
            :
        asm("    LDA.W   A, DP:$$", idata);
            :
        asm("    LDA.W   A, DP:$$", a[2]);
            :
        asm("    LDA.W   A, DP:$$", s.i);
          (Remainder omitted)
            :
}
```

Figure B.19 Description example for specifying DP offset

Figure B.30 shows an example of referencing an auto variable and the result of compiling.

●C source file

```
void near func()
{
        int idata;                                  ←auto variable(DP offset=1)
        asm("     LDA.W   A,DP:$$", idata);
        asm("     CMP.W   A, #1000H ");

   (remainder omitted)
       :
}
```

●Assembly language source file(result of compile)

```
;###    FUNCTION func
;###        FRAME AUTO  (   idata)  size   2,   offset 1
       :
    (abbreviated)
       :
;### C_SRC :      asm("     LDA.W   A,DP:$$", idata);

;#### ASM START
       LDA.W   A,DP:1
    .cline  6                         ←Transfer DP offset 1 to A  register
;### C_SRC :      asm("     CMP.W   A, #1000H ");
       CMP.W   A, #1000H

;#### ASM END

   (remainder omitted)
       :
```

Figure B.30 Example of Reference of auto Variable

Always use this function when referencing an auto variable in an asm function. If you do not use this function, no memory may be allocated for the auto variable. There is also a risk that the program will not run properly after future version upgrades.

You can also use the format shown in Figure B.31 so that auto variables in an asm function use a bit field.

```
asm("        ope-code     $b", bit field name);
```

Figure B.31 Format for Specifying Bit Position

You can only specify one variable name using this format. Figure B.32 is an example.

```
void
func(void)
{
        struct TAG{
                char bit0:1;
                char bit1:1;
                char bit2:1;
                char bit3:1;
        } s;
        asm("seb     $b",s.bit1);
}
```

Figure B.32 Example for Specifying Bit Position

Figure B.33 show examples of referencing auto area bit fields and the results of compiling.

When referencing a bit field in the auto area, you must confirm that it is located within the range that can be referenced using bit operation instructions.

●C source file

```
void
func(void)
{
    struct TAG{
          char bit0:1;
          char bit1:1;
          char bit2:1;
          char bit3:1;
    }s;
    asm("seb    $b",s.bit1);
}
```

●Assembly language source file(compile result)

```
;## #   FUNCTION func
;## #   FRAME   AUTO    (       s)      size   1,       offset 1
;## #   ARG Size(0)     Auto Size(1)    Context Size(5)

        .section        program_F
        .source bit.c
        .cline  3
        .DT     __DT
        .DP     OFF
        .func   _func
        .pub    _func
_func:
        phd
        pht
        tsa
        tad
        .cline  10
;## ASM START
seb #02H,DP:1   ;  s
;## ASM END
        .cline  11
        sem
        pla
        clm
        pld
        rtl
```

Figure B.33 Example of Referencing auto Area Bit Field

B.2.4 Specifying Register Name of register Variable

Register class variables (including parameters) written in C are managed in the registers. You can use the format shown in Figure B.34 to use register variables in asm functions.*1

```
asm("     ope-code     $$, #00H", register Variable name);
```

Figure B.34 Description Format for Register Variables

In NC77, register variables used within functions are managed dynamically. At anyone position, the register used for a register variable is not necessarily always the same one. Therefore, if a register is specified directly in an asm function, it may after compiling operate differently. We therefore strongly suggest using this function to check the register variables.

You can only specify one variable name using this format.
Figure B.35 show examples of referencing register variables and the results of compiling.

●C source file

```
void
func(void)
{
    register int i = 1;

    asm("lda.W  $$,#0000H", i);
}
```

●Assembly language source file(compile result)

```
;## #   FUNCTION func
;## #   ARG Size(0)     Auto Size(0)    Context Size(3)

        .section        program_F
        .source reg.c
        .cline  3
        .DT     __DT
        .DP     OFF
        .func   _func
        .pub    _func
_func:
        .cline  4
        lda.W   A,#0001H     ;  i
        .cline  6
;## ASM START
lda.w   A,#0000H        ;  i
;## ASM END
```

Figure B.35 An Example for Referencing a Register Variable and its Compile Result

* 1 To enable the register modifier, specify option -fenable_register (-fER) when compiling.

B.2.5 Specifying Symbol Name of extern and static Variable

extern and static storage class variables written in C are referenced as symbols.

You can use the format shown in Figure B.36 to use extern and static variables in asm functions.

```
asm("     op-code     A, <DT: or LG:> $$", extern variable name);
asm("     op-code     A, <DT: or LG:> $$", static variable name);
```

Figure B.26 Description Format for Specifying Symbol Name

Only one variable name can be specified by using this description format. The following types are supported for variable names:

- Variable name
- Array name [integer]
- Struct name, member name

```
int idata;
int a[3];
struct TAG{
        int  i;
        int  k;
} s;
void func()
{
            :
        asm("     LDA.W    A, DT:$$", idata );
            :
        asm("     LDA.W    A, DT:$$", a[2] );
            :
        asm("     LDA.W    A, DT:$$", s.i );
          (Remainder omitted)
            :
}
```

Figure B.37 Description example for specifying symbol name

See Figure B.38 for examples of referencing extern and static variables.

● C source file

```
extern far  int ext_val;                    ⇐extern variable

func()
{
        static near  int s_val;             ⇐static variable

        asm("   lda.w   A,LG:$$",ext_val);
        asm("   lda.w   B,DT:$$",s_val);
}
```

● Assembly language source file(compile result)

```
        .pub    _func
 _func:
        .cline  8
 ;## ASM START
        lda     A,LG:_ext_val
        .cline  9
        lda     B,DT:___S0_s_val
 ;## ASM END
        .cline  10
        rtl
        .endfunc        _func

        .SECTION        bss_NE
 ___S0_s_val:           ;### C's name is s_val
        .blkb   2
        .END
```

Figure B.38 Example of Referencing extern and static Variables

You can use the format shown in Figure B.39 to use 1-bit bit fields of extern and static variables in asm functions.

```
        asm("        op-code     $b", bit field name);
```

Figure B.39 Format for Specifying Symbol Names

You can specify one variable name using this format. See Figure B.40 for an example.

```
struct TAG{
        char bit0:1;
        char bit1:1;
        char bit2:1;
        char bit3:1;
} s;

void
func(void)
{
        asm("   bset $b",s.bit1);
}
```

Figure B.30   Example of Specifying Symbol Bit Position

Figure B.31 shows the results of compiling the C source file shown in Figure B.30.

```
;## #   FUNCTION func
;## #   ARG Size(0)     Auto Size(0)    Context Size(3)

        .section        program_F
        .source    kk.c
        .cline  10
        .DT     __DT
        .DP     OFF
        .func   _func
        .pub    _func
_func:
        .cline  11
;## ASM START
seb #02H,DT:_s
;## ASM END
        .cline  12
        rtl
        .endfunc        _func

        .SECTION        bss_NO
        .pub    _s
_s:
        .blkb   1
```

Figure B.31   Example of Referencing Bit Field of Symbol

When referencing the bit fields of extern or static variables, you must confirm that they are located within the range that can be referenced directly using bit operation instructions.

B.2.6 Selectively suppressing optimization

In Figure B.42, the dummy asm function is used to selectively suppress optimization.

```
#pragma ADDRESS port 02H

struct port{
    char bit0:1;
    char bit1:1;
    char bit2:1;
    char bit3:1;
    char bit4:1;
    char bit5:1;
    char bit6:1;
    char bit7:1;
}port;

func()
{
    port.bit0 = 0x01;
    port.bit1 = 0x01;
}
```

Optimization results in any steps to set the two port bits separately being combined as one step.

Optimization ➡

```
seb.B    #03H,_DT:port
```

```
    port.bit0 = 0x01;
    asm();      /* dummy */
    port.bit1 = 0x01;
```

Optimization ➡

Optimization is suppressed.

```
seb.B    01H,_DT:port
seb.B    02H,_DT:port
```

Figure B.42 Example of Suppressing Optimization by Dummy asm

B.2.7 Notes on the asm Function

a. Extended Features Concerning asm functions

When using the asm function [*1] for the following processing, be sure to use the format shown in the coding examples.

(1)Do not specify auto variables or parameters, or 1-bit bit fields using the offset from the direct page register (DPR). Use the format shown in Figure B.43 to specify auto variables and parameters.

```
asm("LDA.W    #01H,$$",i);      ⇐Format for referencing auto variables
asm("SEB      $$", s.bit0);     ⇐Format for checking auto bit fields
```

Figure B.43 Example Coding of asm Function (1/2)

(2)You can specify the register storage class in NC77. When register class variables are compiled with option -fenable_register (-fER), use the format shown in Figure B.44 for register variables in asm functions.

```
asm("LDA.W     $$,#0H", i); ⇐Format for checking register variables
```

Figure B.44 Example Coding of asm Function (2/2)

Note that, when you specify option -O, -OR, or -OS, parameters passed via the registers may, to improve code efficiency, be processed as register variables rather than being moved to the auto area. In this case, when parameters are specified in an asm function, the assembly language is output using the register names instead of the variable's DPR offset.

(3)Do not use the CLM, CLP, SEM or SEP instructions, etc., in the asm function to change the m or x flags in the processor status register. See Figure B.45 for the format for changing the m and x flags.

```
asm("CLP      M, X");      ←Do not use this format.
      ▼
asm(0, 0);              ←Format for specifying asm (value of m flag, value of x flag)
```

Figure B.45 Example Coding of asm Function (1)

*1. In this manual, we refer to subroutines written in assembly language as assembler functions. Those written using asm() in a C program are called asm functions or inline assembler.

b. Notes on DT register

(1)When the content of the data bank register (DT) is changed by the asm function, include the code shown in Figure B.46 at the end of that asm function to return the DT to its original state.

```
asm("   LDT    #1");                    ←DTchanged
asm("   .DT    1");
asm("   LDA    A,DT:TABLE");
               :
         (abbreviated)
               :
asm("   LDT    #__DT");                 ←DT returned to original state
```

Figure B.46 Restoring Data Bank Register

c. Notes on Labels

The assembler source files generated by NC77 include internal labels in the format shown in Figure B.47. Therefore, you should avoid using labels in an asm function that might result in duplicate names.

● Labels consisting of one uppercase letter and one or more numerals

Examples: A1:
C9877:

● Labels consisting of two or more characters preceded by the underscore (_)

Examples: __LABEL:
___START:

Figure B.47 Label Format Prohibited in asm Function

d. Notes on Comments in Assembler Code

The assembler source files generated by NC77 contain comments following the rules in Figure B.48. Therefore, you should avoid including comments that adhere to the same rules in an asm function.

● ;(semicolon) and two pound signs (##): Control information for compiler

Example: ;## Comment

● ;(semicolon) and three pound signs (###): Control information for debugger

Example: ;### Comment

● ;(semicolon) and four pound signs (####): Control information for loop77

Example: ;#### Comment

Figure B.48 Rules for Comments in Assembly Language Source Files

B.3 Description of Japanese Characters

NC77 allows you to include Japanese characters in your C source programs. This chapter describes how to do so.

B.3.1 Overview of Japanese Characters

In contrast to the letters in the alphabet and other characters represented using one byte, Japanese characters require two bytes. NC77 allows such 2-byte characters to be used in character strings, character constants, and comments. The following character types can be included:

- kanji
- hiragana
- full-size katakana
- half-size katakana

Only the following kanji code systems can be used for Japanese characters in NC77.

- EUC (excluding user-defined characters made up of 3-byte code)
- Shift JIS (SJIS)

B.3.2 Settings Required for Using Japanese Characters

The following environment variables must be set in order to use kanji codes:

- Environment variable specifying input code system .... NCKIN
- Environment variable specifying output code system .... NCKOUT

Figure B.49 is an example of setting the environment variables.

```
[UNIX]
This example sets the input to EUC codes and the output to Shift JIS codes.
% setenv NCKIN EUC
% setenv NCKOUT SJIS

[MS-Windows]
Include the following in your autoexec.bat file:
set NCKIN=SJIS
set NCKOUT=SJIS
```

Figure B.49 Example Setting of Environment Variables NCKIN and NCKOUT

In NC77, the input kanji codes are processed by the cpp77 preprocessor. cpp77 changes the codes to EUC codes. In the last stage of token analysis in the ccom77 compiler, the EUC codes are then converted for output as specified in the environment variable.

B.3.3 Japanese Characters in Character Strings

Figure B.50 shows the format for including Japanese characters in character strings.

```
L" 漢字文字列 "
```

Figure B.50   Format of Kanji code Description in Character Strings

If you write Japanese using the format " 漢字文字列 " as with normal character strings, it is processed as a pointer type to a char type when manipulating the character string. You therefore cannot manipulate them as 2-byte characters.

To process the Japanese as 2-byte characters, precede the character string with L and process it as a pointer type to a wchar_t type. wchar_t types are defined (typedef) as unsigned short types in the standard header file stdlib.h.

Figure B.51 shows an example of a Japanese character string.

```
#include <stdlib.h>

void near func()
{
        wchar_t JC[4] = L" 文字列 ";        ←[1]

         (remainder omitted)
            :
```

Figure B.51   Example of Japanese Character Strings Description

Figure B.52 is a memory map of the character string initialized in (1) in Figure B.39.

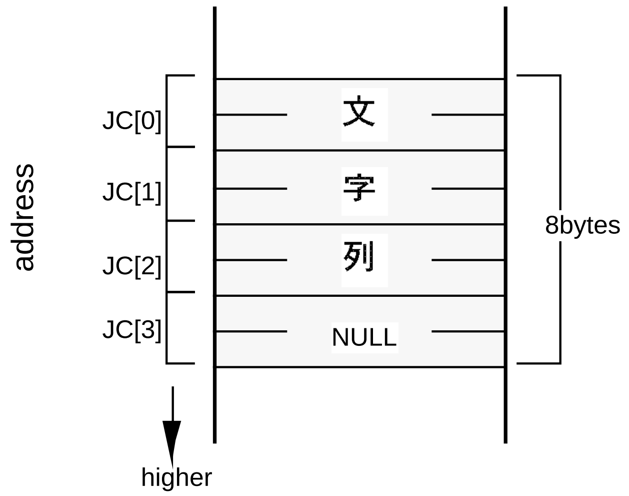


Figure B.52   Memory Location of wchar_t Type Character Strings

B.3.4 Using Japanese Characters as Character Constants

Figure B.53 shows the format for using Japanese characters as character constants.

```
L'漢 '
```

Figure B.53   Format of Kanji code Description in Character Strings

As with character strings, precede the character constant with L and process it as a wchar_t type. If, as in '文字 ', you use two or more characters as the character constant, only the first character "文 " becomes the character constant.

Figure B.54 shows examples of how to write Japanese character constants.

```
#include <stdlib.h>

void near func()
{
        wchar_t JC[5];

        JC[0] = L'文 ';
        JC[1] = L'字 ';
        JC[2] = L'定 ';
        JC[3] = L'数 ';

          (remainder omitted)
              :
```

Figure B.54   Format of Kanji Character Constant Description

Figure B.55 is a memory map of the array to which the character constant in Figure B.42 has been assigned.

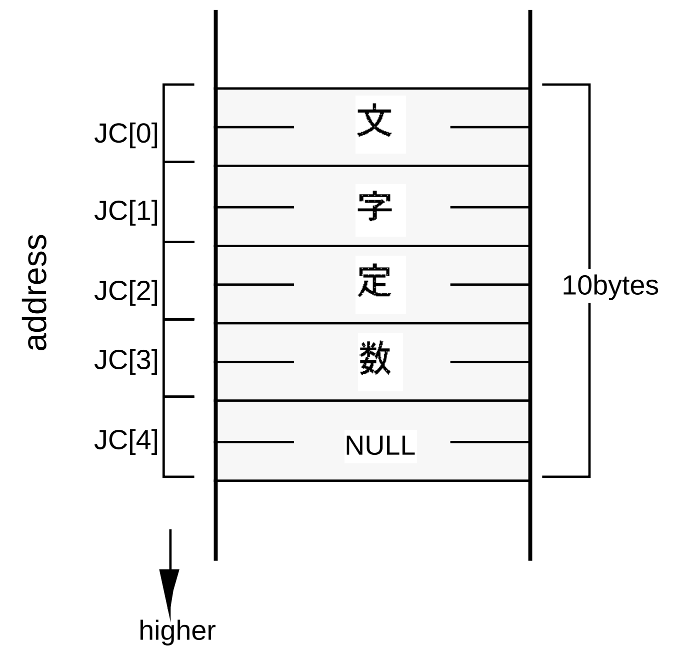


Figure B.55   Memory Location of wchar_t Type Character Constant Assigned Array

B.4 Default Argument Declaration of Function

NC77 allows you to define default values for the arguments of functions in the same way as with the C++ facility. This chapter describes NC77's facility to declare the default arguments of functions.

B.4.1 Overview of Default Argument Declaration of Function

NC77 allows you to use implicit arguments by assigning parameter default values when declaring a function's prototype. By using this facility you can save the time and labor that would otherwise be required for writing frequently used values when calling a function.

B.4.2 Format of Default Argument Declaration of Function

Figure B.56 shows the format used to declare the default arguments of a function.

```
Storage class specifierΔ Type declaratorΔ Declarator([Dummy argument[=Default value
or variable],...]);
```

Figure B.56 Format for declaring the default arguments of a function

Figure B.57 shows an example of declaration of a function, and Figure B.58 shows a result of compiling of sample program which shows at Figure B.57.

```
extern  int  func(int i=1, int j=2);
                :
        (abbreviated)
```

Figure B.57 Example for declaring the default arguments of a function

```
_main:
      .cline    5
;## # C_SRC :                func();
      pea             #0002H            ⇐ second argument   : 2
      lda.W           A,#0001H          ⇐ first argument    : 1
      jsrl            ?func
      plx
      .cline    6
;## # C_SRC :                func(3);
      pea             #0002H            ⇐second argument    : 2
      lda.W           A,#0003H          ⇐first argument     : 3
      jsrl            ?func
      plx
      .cline    7
;## # C_SRC :                func(3,5);
      pea             #0005H            ⇐second argument    : 5
      lda.W           A,#0003H          ⇐first argument     : 3
      jsrl            ?func
      plx
      .cline    8
;## # C_SRC :         }
      rtl
             :
        (omitted)
             :
```

Note) In NC77, arguments are stacked in revere order beginning with the argument that is declared last in the function. In this example, arguments are passed via registers as they are processed.

Figure B.58 Compiling Result of smp1.c(smp1.a77)

A variable can be written for the argument of a function.

Figure B.59 shows an example where default arguments are specified with variables. Figure B.60 shows a compile result of the sample program shown in Figure B.59.

```
int  near  sym ;
int  func( int  i = sym);                ⇐ Default argument is specified with a variable.

void  main(void)
{
      func();                            ⇐ Function is called using variable (sym) as argument.
}
              :
          (omitted)
              :
```

Figure B.59 Example for specifying default argument with a variable (smp2.c)

```
_main:
      .cline    6
;## # C_SRC :                func();
      lda       A,DT:_sym             ⇐ Function is called using variable (sym) as argument.
      jsrl      ?func
      .cline    7
;## # C_SRC :         }
      rtl
```

Figure B.60 Compile Result of smp2.c (smp2.a77)

When declaring the default argument of a function, pay attention to the following:

(1) When specifying a default value for multiple arguments

When specifying a default value in a function that has multiple arguments, always be sure to write values beginning with the last argument. Figure B.61 shows examples of incorrect description.

```
void  func1(int i, int  j=1, int  k=2);          /* correct */

void  func2(int  i, int  j, int  k=2);           /* correct */

void  func3( int i = 0, int  j, int  k);         /* incorrect */

void  func4(int  i = 0, int  j, int  k = 1);     /* incorrect */
```

Figure B.61 Examples of Prototype Declaration

(2) When specifying a variable for a default value

When specifying a variable for a default value, write the prototype declaration of a function after declaring the variable you specify. If a variable is specified for the default value of an argument that is not declared before the prototype declaration of a function, it is processes as an error.

B.4.3 Restrictions on Default Argument Declaration of Function

The default argument declaration of a function is subject to some restrictions as listed below. These restrictions must be observed.

- When there are multiple arguments of a function when specifying default arguments, always be sure to write them sequentially one argument after another.
- Variables can be specified for default arguments. However, when specifying a variable, be sure to declare the default arguments of a function after declaring the variable you want to specify. If you specify for the default value of an argument any variable that is not declared yet when the default arguments of a function are declared, your program may result in an error when compiled.

B.5 inline Function Declaration

NC77 allows you to specify the inline storage class in the similar manner as in C++. By specifying the inline storage class for a function, you can expand the function inline.

This chapter describes specifications of the inline storage class.

B.5.1 Overview of inline Storage Class

The inline storage class specifier declares that the specified function is a function to be expanded inline. The functions specified as belonging to the inline storage class are defined as macros at the assembly language level.

B.5.2 Declaration Format of inline Storage Class

The inline storage class specifier must be written in a syntactically similar format to that of the static and extern-type storage class specifiers when declaring the inline storage class. Figure B.62 shows the format used to declare the inline storage class.

```
inline∆type specifier∆function;
```

Figure B.62 Declaration Format of inline Storage Class

Figure B.63 shows an example of declaration of a function.

```
extern int  i;
inline  int  func(void);            ⇐Prototype declaration of function
{
        i++;
}
void main()
{
      int  s;;
      s = func();                   ⇐Definition of body of function
}
```

Figure B.63 Example for Declaring inline Storage Class

```
;## #     FUNCTION func
;## #     ARG Size(0) Auto Size(0)      Context Size(0)

    .source test.c
    .section    program_F
;## # C_SRC :   {
    .DT   __DT
    .DP   OFF
_func:    .MACRO
;## # C_SRC :         return ++i;
    inc   DT:_i
    lda   A,DT:_i
    .ENDM

;## #     FUNCTION main
;## #     FRAME AUTO  (       s)  size   2,   offset 1
;## #     ARG Size(0) Auto Size(2)      Context Size(5)

;## # C_SRC :   {
    .DT   __DT
    .DP   OFF
    .func _main
    .pub  _main
_main:
    phd
    pha
    tsa
    tad
    .cline      11
;## # C_SRC :         s = func();
    _func
    sta   A,DP:1      ;  s
    .cline      12
;## # C_SRC :   }
    plx
    pld
    rtl
    .endfunc    _main
    .END
```

Figure B.64   Compile Result of sample program (smp.a77)

```
                     ;## # FUNCTION main
                     ;## # FRAME AUTO  (       s)  size   2,   offset 1
                     ;## # ARG Size(0) Auto Size(2)      Context Size(5)

                     .DT   __DT
                     .DP   OFF
                     .func _main
                     .pub  _main
000000               _main:
000000  0B                 phd
000001  48                 pha
000002  3B                 tsa
000003  5B                 tad
                     .cline      11
000004               _func
000004  EE0000  E +  inc   DT:_i
000007  AD0000  E +  lda   A,DT:_i
                +    .ENDM
00000A  8501               sta   A,DP:1      ;  s
                     .cline      12
00000C  FA                 plx
00000D  2B                 pld
00000E  6B                 rtl
                     .endfunc    _main
```

Figure B.65   Macro-expansion result (smp.prn) of smp.a77

B.5.3 Restrictions on inline Storage Class

When specifying the inline storage class, pay attention to the following :

(1) Regarding the nesting of inline functions

You can call another inline function from an inline function. However, since an inline function uses a macro, this is subject to restrictions depending on the number of levels macros can be nested in the assembler. Refer to the RASM77 User's Manual for details about the number of nested levels of macros. Note also that no inline function can be recursive call.

(2) Regarding the definition of an inline function

When specifying inline storage class for a function, be sure to define the body of the function in addition to declaring it. Make sure that this body definition is written in the same file as the function is written . The description in Figure B.66 is processed as an error in NC77.

```
inline void func(int i);

void main( void )
{
        func(1);
}
```

[Error Message]

```
[Error(ccom):smp.c,line 5] inline function's body is not declared previously
   ==⇒          func(1);
Sorry, compilation terminated because of these errors in main().
```

Figure B.66 Example of inappropriate code of inline function (1)

Furthermore, after using some function as an ordinary function if you define that function as an inline function later, your inline specification is ignored and all functions are handled as static functions. In this case, NC77 generates a warning. (See Figure B.67.)

```
int func(int i);

void main( void )
{
        func(1);
}

inline int func(int i)
{
        return i;
}
```

[Warning Message]

```
[Warning(ccom):smp.c,line 9] inline function is called as normal function before
,change to static function.
   ==⇒ {
```

Figure B.67 Example of inappropriate code of inline function (2)

(3) Regarding the address of an inline function

Since an inline function is a macro definition, the function itself does not have an address. Therefore, if the & operator is used for an inline function, the software assumes an error. (See Figure B.68.)

```
int func(int i)
{
        return i;
}

main()
{
        int (*f)(int);

        f = &func;
}
```

[Error Message]

```
[Error(ccom):smp.c,line 10] can't get inline function's address by '&' operator
   ==⇒         f = &func;
Sorry, compilation terminated because of these errors in main().
```

Figure B.68 Example of inappropriate code of inline function (3)

(4) Declaration of static data

If static data is declared in an inline function, the body of the declared static data is allocated in units of files. For this reason, if an inline function consists of two or more files, this results in accessing different areas. Therefore, if there is static data you want to be used in an inline function, declare it outside the function. If a static declaration is found in an inline function, NC77 generates a warning. Mitsubishi does not recommend entering static declarations in an inline function. (See Figure B.69.)

```
inline int func( int j)
{
        static int i = 0;

        i++;
        return i + j;
}
```

[Warning Message]

```
[Warning(ccom):smp.c,line 3] static valuable in inline function
   ==⇒         static int i = 0;
```

Figure B.69 Example of inappropriate code of inline function (4)

(5) Regarding debug information

NC77 does not output C language-level debug information for inline functions. Therefore, you need to debug inline functions at the assembly language level.

B.6 Extension of Comments

NC77 allows comments enclosed between "/*" and "*/" as well as C++-like comments starting with "//".

B.6.1 Overview of "//" Comments

In C, comments must be written between "/*" and "*/". In C++, anything following "//" is treated as a comment.

B.6.2 Comment "//" Format

When you include "//" on a line, anything after the "//" is treated as a comment.
Figure B.70 shows comment format.

```
        //    comments
```

Figure B.70   Comment Format

Figure B.71 shows example comments.

```
void
func(void)
{
    int i;      /* This is commentes */
    int j;      // This is commentes
      :
      :
}
```

Figure B.71   Example Comments

B.7 #pragma Extended Functions

B.7.1 Index of #pragma Extended Functions

Following index tables show contents and formation for #pragma extended functions.

a. Using Memory Mapping Extended Functions

Table B.3 Memory Mapping Extended Functions

Extented function	Description
#pragma ROM	Maps the specified variable to rom Syntax : `#pragma ROM` *variable_name* Example : `#pragma ROM val` ❋The variable normally must be located in the rom section using the const qualifier.
#pragma SECTION	Changes the section name generated by NC77 Syntax : `#pragma SECTION` *section_name new_section_name* Example : `#pragma SECTION bss nonval_data`
#pragma STRUCT	1. Inhibits the packing of structures with the specified tag Syntax : `#pragma STRUCT` *structure_tag* `unpack` Example : `#pragma STRUCT TAG1 unpack` 2. Arranges members of structures with the specified tag and maps even sized members first Syntax : `#pragma STRUCT` *structure_tag* `arrange` Example : `#pragma STRUCT TAG1 arrange`

b. Using Extended Functions for Target Devices

Table B.4 Extended Functions for Use with Target Devices

Extended function	Description
#pragma ADDRESS (#pragma EQU)	Specifies the absolute address of a variable. For near variables, this specifies the address within the bank. Syntax : `#pragma ADDRESS`Δ*variable-name*Δ*absolute-address* Example : `#pragma ADDRESS port0 2H` ❋#pragma EQU can also be used for maintaining compatibility with C77.
#pragma INTERRUPT (#pragma INTF)	Declares an interrupt handling function written in C language. This declaration causes code to perform a procedure for the interrupt handling function to be generated at the entry or exit to and from the function. Syntax : `#pragma INTERRUPT`Δ`[/E]`Δ*interrupt-handling-function-name* Example : `#pragma INTERRUPT int_func` Example : `#pragma INTERRUPT /E int_func` ❋ #pragma INTF can also be used for maintaining compatibility with C77.
#pragma PARAMETER	Declares that, when calling an assembler function, the parameters are passed via specified registers. Syntax : `#pragma PARAMETER`Δ*function_name* (*register_name*) Example : `#pragma PARAMETER asm_func(A,X)` ❋Always be sure to declare the prototype of the function before entering this declaration.

c. Using MR7700 Extended Functions

Table B.5 Extended Functions for MR7700

Extended function	Description
#pragma ALMHANDLER	Declares the name of the MR7700 alarm handler function Syntax : `#pragma ALMHANDLER` *function-name* Example : `#pragma ALMHANDLER alm_func`
#pragma CYCHANDLER	Declares the name of the MR7700 cycle start handler function Syntax : `#pragma CYCHANDLER` *function-name* Example : `#pragma CYCHANDLER cyc_func`
#pragma INTHANDLER #pragma HANDLER	Declares the name of the MR7700 interrupt handler function Syntax1 : `#pragma INTHANDLER` *function-name* Syntax2 : `#pragma HANDLER` *function-name* Example : `#pragma INTHANDLER int_func`
#pragma TASK	Declares the name of the MR7700 task start function Syntax : `#pragma TASK` *task-start-function-name* Example : `#pragma TASK task1`

Supplement: The above extended function normally is generated by the configurator, so that the user need not be concerned with it.

d. DT Register Operation Extended Function

Table B.6 DT Register Operation Extended Function

Extended function	Content and related rules of function
#pragma LOADDT	Specifies the function that loads the DT value at the beginning of the function when compiling. 1.Only functions for which the function is coded after #pragma LOADDT are valid. 2.No processing occurs if you specify other than a function name. 3.No error occurs if you duplicate #pragma LOADDT declarations.

e. Function Call Extended Function

Table B.7 Extended Functions for Function Calls

Extended function	Content and related rules of function
#pragma M1FUNCTION	Calls a function in which M flag = 1. 1.Specify the same settings for the M and X flags for both calling and called functions.

f. The Other Extensions

Table B.8 Using Inline Assembler Description Function

Extended feature	Description

#pragma ASM
#pragma ENDASM
	Specifies an area in which statements are written in assembly language.
	Syntax : `#pragma ASM`
	`#pragma ENDASM`
	Example : `#pragma ASM`
	`lda.w A,#00H`
	`adc.w A,#02H`
	`#pragma ENDASM`

#pragma PAGE	Indicates a new-page point in the assembler listing file. Syntax : `#pragma PAGE` Example : `#pragma PAGE`

B.7.2 Using Memory Mapping Extended Functions

NC77 includes the following memory mapping extended functions.

#pragma ROM

Map to rom section

Function : Maps specified data (variable) to rom section

Syntax : #pragma ROMΔ*variable_name*

Description : This extended function is valid only for variables that satisfy one or other of the following conditions:

[1] Non-extern variables defined outside a function (Variables for which an area is secured)
[2] Variables declared as static within the function

Rules :
1. If you specify other than a variable, it will be ignored.
2. No error occurs if you specify #pragma ROM more than once.
3. The data is mapped to a rom section with initial value 0 if you do not include an initialization expression.

Example :

[C language source program]

```
#pragma ROM i
unsigned int    i;              ⇐Variable i, which satisfies condition[1]

void func()
{
    static int i = 20;          ⇐Variable i, which satisfies condition[2]
         :
    (remainder omitted)
```

[Assembly language source program]

```
    .section    rom_NE      ;### C's name is i      ⇐Variable i, which satisfies
___S0_i:                                                condition[2]
    .word   0014H
    .glb    _i
_i:                     ⇐Variable i, which satisfies condition[1]
    .byte   00H
    .byte   00H
```

Figure B.72 Example Use of #pragma ROM Declaration

Note: The variable normally must be located in the rom section using the const modifier.

#pragma SECTION

Change section name

Function : Changes the names of sections generated by NC77

Syntax : #pragma SECTIONΔ*section name*Δ*new section name*

Description : Specifying the program section in a #pragma SECTION declaration changes the section names of all subsequent functions.
Specifying a data section (data, bss and rom) in a #pragma SECTION declaration changes the names of all data sections defined in that file.
If you need to add or change section names after using this function to change section names, change initialization, etc., in the startup program for the respective sections.

Example :

[C source program]

```
#pragma SECTION program pro1   ⇐Changes name of program section to pro1
void func( void );
         :
    (remainder omitted)
```

[Assembly language source program]

```
;###    FUNCTION func

    .section    pro1          ⇐Maps to pro1 section
    ._file      'smp.c'
    ._line      9
    .pub        _func
_func:
```

Figure B.73 Example Use of #pragma SECTION Declaration

Note : When modifying the name of a section except "interrupt", note that the section's location attribute (e.g., _NE or _INE) is added after the section name.

#pragma STRUCT

Control structure mapping

Function : [1] Inhibits packing of structures
[2] Arranges structure members

Syntax : [1] #pragma STRUCTΔ*structure_tag*Δunpack
[2] #pragma STRUCTΔ*structure_tag*Δarrange

Description and Examples : In NC77, structures are packed. For example, the members of the structure in Figure B.74 are arranged in the order declared without any padding.

```
struct s {
        int     i;
        char    c;
        int     j;
};
```

Member name	Type	Size	Mapped location (offset)
i	int	16 bits	0
c	char	8 bits	2
j	int	16 bits	3

Figure B.74 Example Mapping of Structure Members (1)

[1]Inhibiting packing

This NC77 extended function allows you to control the mapping of structure members. Figure B.75 is an example of mapping the members of the structure in Figure B.64 using #pragma STRUCT to inhibit packing.

```
struct s {
        int     i;
        char    c;
        int     j;
};
```

Member name	Type	Size	Mapped location (offset)
i	int	16 bits	0
c	char	8 bits	2
j	int	16 bits	3
Padding	(char)	8 bits	–

Figure B.75 Example Mapping of Structure Members (2)

As shown Figure B.75, if the total size of the structure members is an odd number of bytes, #pragma STRUCT adds 1 byte as packing after the last member. Therefore, if you use #pragma STRUCT to inhibit padding, all structures have an even byte size.

Description : [2]Arranging members

This NC77 extended function allows you to map the all odd-sized structure members first, followed by even-sized members. Figure B.76 shows the offsets when the structure shown in Figure B.74 is arranged using #pragma STRUCT.

```
struct s {
        int     i;
        char    c;
        int     j;
};
```

Member name	Type	Size	Mapped location (offset)
i	int	16 bits	0
j	int	16 bits	2
c	char	8 bits	4

Figure B.76 Example Mapping of Structure Members (3)

You must declare #pragma STRUCT for inhibiting packing and arranging the structure members before defining the structure members.

Examples :

```
#pragma STRUCT  TAG   unpack
struct TAG {
        int     i;
        char    c;
} s1;
```

Figure B.77 Example of #pragma STRUCT Declaration

B.7.3 Using Extended Functions for Target Devices

NC77 includes the following extended functions for target devices.

#pragma ADDRESS (#pragma EQU)

Specify absolute address of I/O variable

Function : Specifies the absolute address of a variable. For near variables, the specified address is within the bank.

Syntax : #pragma ADDRESSΔ*variable-name*;*absolute-address*

Description : The absolute address specified in this declaration is expanded as a character string in an assembler file and defined in pseudo instruction .EQU. The format for writing the numerical values therefore depends on the assembler, as follows:

- Append 'B' or 'b' to binary numbers
- Append 'O' or 'o' to octal numbers
- Write decimal integers only.
- Append 'H' or 'h' to hexadecimal numbers. If the number starts with letters A to F,precede it with 0.

Rules :

1. All storage classes such as extern and static for variables specified in `#pragma ADDRESS` are invalid.
2. Variables specified in `#pragma ADDRESS` are valid only for variables defined outside the function.
3. `#pragma ADDRESS` is valid for previously declared variables.
4. `#pragma ADDRESS` is invalid if you specify other than a variable.
5. No error occurs if a `#pragma ADDRESS` declaration is duplicated, but the last declared address is valid.
6. An error occurs if you include an initialization expression.
7. A #pragma ADDRESS declaration does not have the same effect as the near attribute. If the near area is mapped to bank 1 (compiled with the nc77 command line option -bank=1), compiling the example program in Figure B.68 results in the absolute address of " io " being 10024H. This is because variable io has the near attribute and because the DT value is added.

Example :

```
#pragma ADDRESS io      24H
int near        io;

func()
{
        io = 10;
}
```

Figure B.68 #pragma ADDRESS Declaration and near/far Attribute (1)

Note : For compatibility with C77 versions prior to V.2.10 before can accept files that include `#pragma EQU`. The absolute address using this format is written using the C conventions.

Rule: When specifying an absolute address that is not dependent on the DT value, specify far when declaring the variable.

```
#pragma ADDRESS io      24H
int far         io;

func()
{
         io = 10;
}
```

Figure B.79 #pragma ADDRESS Declaration and near/far Attribute (2)

Example:

```
#pragma ADDRESS port0   2H
#pragma ADDRESS port1   3H
#pragma ADDRESS p0d     4H
#pragma ADDRESS p1d     5H
char    port0, port1, p0d, p1d;

void main()
{
        p0d = p1d = 0xFF;
        port0 = 0xAA;
        port1 ^= port0;
}
```

Figure B.80 Example #pragma ADDRESS Declaration

#pragma INTERRUPT (#pragma INTF)

Declare interrupt function

Function : Declares an interrupt handler

Syntax : #pragma INTERRUPTΔ[/E]Δ*interrupt-handler-name*

Description : By using the above format to declare interrupt processing functions written in C, NC77 generates the code for performing the following interrupt processing at the entry and exit points of the function.

- In entry processing, all registers of the 7700 family are saved to the stack.
- In exit processing, the saved registers are restored and control is returned to the calling function by the RTI instruction.
- Functions declared in #pragma INTERRUPT are mapped to the interrupt section.

You may specify /E in this declaration:

/E : Multiple interrupts are enabled immediately after entering the interrupt. This improves interrupt response.

Rules :
1. A warning is output when compiling if you declare interrupt processing functions that take parameters
2. A warning is output when compiling if you declare interrupt processing functions that return a value. Be sure to declare that any return value of the function has the void type.
3. Only functions for which the function is defined after a `#pragma INTERRUPT` declaration are valid.
4. No processing occurs if you specify other than a function name.
5. No error occurs if you duplicate `#pragma INTERRUPT` declarations.
6. Do not declare near or far for functions declared in `#pragma INTERRUPT`.

Example :

```
#pragma INTERRUPT /E i_func


void i_func()
{
        int_counter += 1;
}
```

Figure B.81 Example of #pragma INTERRUPT Declaration

Note : For compatibility with C77 versions prior to V.2.10 before can accept files that include `#pragma INTF`.

#pragma PARAMETER

Declare assembler function that passed arguments via register

Function : Declares an assembler function that passes parameters via registers

Syntax : #pragma PARAMETER∆*assembler-function-name* (*register-name*, *register-name*,…)

Description : This extended function declares that, when calling an assembler function, its parameters are passed via registers.

● float types, long types, far pointer types (16-bit register-pairs)	: AB and XY
● int types, near pointer types (16-bit registers)	: A, B, X, and Y
● char types (8-bit registers)	: A, B, X, and Y

* Register names are NOT case-sensitive.

Rules :

1. Always put the prototype declaration for the assembler function before the `#pragma PARAMETER` declaration. If you fail to make the prototype declaration, a warning is output and `#pragma PARAMETER` is ignored.

2. Follow the following rules in the prototype declaration:

a. Note also that the number of parameters specified in the prototype declaration must match that in the `#pragma PARAMETER` declaration.

b. The following types cannot be declared as parameters for an assembler function in a `#pragma PARAMETER` declaration:

- structure-type and union-type
- double-type

c. The assembler functions shown below cannot be declared:

- Functions returning structure or union type
- double-type

3. An error occurs, when you write the function entity specified in `#pragma PARAMETER` in C language.

Example :

```
int asm_func(unsigned int, unsigned int); ⇐Prototype declaration for
#pragma PARAMETER asm_func(X, Y)            the assembler function

void main()
{
     int     i, j;
     i = 0x7FFD;
     j = 0x007F;

     asm_func( i, j );                    ⇐Calling the assembler function

}
```

Figure B.82 Example of #pragma PARAMETER Declaration

B.7.4 Using MR7700 Extended Functions

NC77 has the following extended functions which support the real-time operating system MR7700.

#pragma ALMHANDLER

Alarm handler declaration

Function : Declares an MR7700 alarm handler

Syntax : #pragma ALMHANDLER∆*alarm-handler-name*

Description : By using the above format to declare an alarm handler (a function) written in C, NC77 generates the code for the alarm handler to be used at the entry and exit points of the function.

- The alarm handler is called from the system clock interrupt by the JSR instruction and returns by the RTS instruction.
- Load the value of the data bank register (DT) during entry processing and restore old DT value at exit processing.

Rules : Functions declared in #pragma ALMHANDLER are mapped to the interrupt section.

1. You canNOT write alarm handlers that take parameters.
2. The return value from the alarm handler must be type void in the declaration.
3. Only the function definition put after `#pragma ALMHANDLER` are valid.
4. No processing occurs if you specify other than a function name.
5. No error occurs if you duplicate `#pragma ALMHANDLER` declarations.
6. A compile error occurs if you use any function specified in one of the following declarations in `#pragma ALMHANDLER`:
    - #pragma INTERRUPT
    - #pragma INTHANDLER
    - #pragma HANDLER
    - #pragma CYCHANDLER
    - #pragma TASK

Example :

```
#include   <mr7700.h>
#include   "id.h"

#pragma ALMHANDLER      alm

void alm()             ⇐Be sure to declare as type void.
{
          :
      (omitted)
          :
}
```

Figure B.83 Example of #pragma ALMHANDLER Declaration

#pragma CYCHANDLER

Cyclic handler declaration

Function : Declares an MR7700 cyclic handler

Syntax : #pragma CYCHANDLER∆*cyclic-handler-name*

Description : By using the above format to declare a cyclic handler (a function) written in C, NC77 generates the code for the cyclic handler to be used at the entry and exit points of the function.

- The cyclic handler is called from the system clock interrupt by the JSR instruction and returns by the RTS instruction.
- Load the value of the data bank register (DT) during entry processing and restore old DT value at exit processing.

Rules : Functions declared in #pragma ALMHANDLER are mapped to the interrupt section.

1. You canNOT write cyclic handlers that take parameters.
2. The return value from the cyclic handler must be type void in the declaration.
3. Only the function definition put after `#pragma CYCHANDLER` are valid.
4. No processing occurs if you specify other than a function name.
5. No error occurs if you duplicate `#pragma CYCHANDLER` declarations.
6. A compile error occurs if you use any function specified in one of the following declarations in `#pragma CYCHANDLER`:
   - #pragma INTERRUPT
   - #pragma INTHANDLER
   - #pragma HANDLER
   - #pragma ALMHANDLER
   - #pragma TASK

Example :

```
#include   <mr7700.h>
#include   "id.h"

#pragma CYCHANDLER      cyc

void cyc()            ⇐Be sure to declare as type void.
{
          :
      (omitted)
          :
}
```

Figure B.84 Example of #pragma CYCHANDLER Declaration

#pragma INTHANDLER (#pragma HANDLER)

Interrupt handler declaration

Function : Declares an MR7700 OS-dependent interrupt handler

Syntax : [1] #pragma INTHANDLERΔ*interrupt-handler-name*
[2] #pragma HANDLERΔ*interrupt-handler-name*

Description : By using the above format to declare an interrupt handler written in C, NC77 generates the code for the following processes to be used at the entry and exit points of the function.

1.Entry processing

- Push the registers to the current stack.
- Save the stack pointer (SP) to the task control block (TCB).
- Switch to the system stack.

2.Exit processing

- Returns from interrupt using ret_int system call. Also returns from interrupt using ret_int system call when returning at a return statement partway through the function.

Using #pragma INTHANDLER declarations, the program format differs from MR7700 (v.2.12) as follows:

- There is no need to call the IntEntry macro at the start of the interrupt handler.
- You can now use storage class AUTO variables in interrupt handlers.
- You can now include complex expressions such as those that use work areas in interrupt handlers.

Functions declared in #pragma INTHANDLER are mapped to the interrupt section.
To declare an MR7700 OS-independent interrupt handler, use #pragma INTERRUPT.

Rules :

1. You canNOT write interrupt handlers that take parameters.
2. The return value from the interrupt handler must be type void in the declaration.
3. Do NOT use the ret_int system calls from C.
4. Only the function definition put after `#pragma INTHANDLER` are valid.
5. No processing occurs if you specify other than a function name.
6. No error occurs if you duplicate `#pragma INTHANDLER` declarations.
7. A compile error occurs if you use any function specified in one of the following declarations in `#pragma INTHANDLER`:
   - #pragma INTERRUPT
   - #pragma HANDLER
   - #pragma ALMHANDLER
   - #pragma CYCHANDLER
   - #pragma TASK

Example :

```
#include    <mr7700.h>
#include    "id.h"

#pragma INTHANDLER      hand

void hand()
{
          :
      (omitted)
          :
    /* ret_int(); */
}
```

Figure B.85   Example of #pragma INTHANDLER Declaration

#pragma TASK

Task start function declaration

Function : Declares an MR7700 task start function

Syntax : #pragma TASKΔ*task-start-function-name*

Description : By using the above format to declare a task start function written in C, NC77 generates the code for the task start function to be used at the entry and exit points of the function.

1.Entry processing
- Does not save old frame pointer (DPR)

2.Exit processing
- Ends at ext_tsk system call. Also returns using ext_tsk system call even when returning at return statement part way through function.

Rules :
1. You need not put the ext_tsk system call to return from the task.
2. The return value from the task must be type void in the declaration.
3. Only the function definition put after `#pragma TASK` are valid.
4. No processing occurs if you specify other than a function name.
5. No error occurs if you duplicate `#pragma TASK` declarations.
6. A compile error occurs if you use any function specified in one of the following declarations in `#pragma TASK`:
   - #pragma INTERRUPT
   - #pragma INTHANDLER
   - #pragma HANDLER
   - #pragma ALMHANDLER
   - #pragma CYCHANDLER

Example :

```
#include   <mr7700.h>
#include   "id.h"

#pragma TASK      main
#pragma TASK      tsk1

void main()                  ⇐Be sure to declare as type void.
{
          :
      (omitted)
          :
       sta_tsk(ID_idle);
       sta_tsk(ID_tsk1);
       /* ext_tsk(); */      ⇐You need not use ext_tsk.
}

void tsk1()
          :
    (remainder omitted)
```

Figure B.86 Example of #pragma TASK Declaration

B.7.5 Using the DT Register Operation Extended Function

NC77 includes the following extended function for operating the direct page register (DT).

#pragma LOADDT

Specify function to load DT register

Function: Specifies the function that loads the DT value at the beginning of the function when compiling

Format: #pragma LOADDTD function_name

Description: By declaring the function in #pragma LOADDT , NC77 generates the code for loading the value of the data bank register (DT) specified in the compiler option -bank= at the beginning of the function.

Rules:

1. Only functions for which the function is coded after #pragma LOADDT are valid.
2. No processing occurs if you specify other than a function name.
3. No error occurs if you duplicate #pragma LOADDT declarations.

This function is used to switch the DT value (the value of the bank in the near area) and improve efficiency in application programs with more than 64KB of data. It is essential to fully understand the relationship between the near and far attributes and the 7700 family addressing modes when using this function.

Example:

[C source program]

```
#pragma LOADDT      func

void func()
{
    int     i, j;
          :
  (remainder omitted)
          :
```

[Assembly language source program]

```
;###    FUNCTION func

    .source sor2.c
    .section    program_F
    .DT __DT
    .DP OFF
    .func   _func
    .pub    _func
_func:
    pht
    ldt #__DT          ←Load DT value when compiling
    .cline  7
          :
  (remainder omitted)
          :
```

Figure B.87 Example Use of #pragma LOADDT Declaration

B.7.6 Using the Function Call Extended Function

NC77 includes the following extended function for function calls.

#pragma M1FUNCTION

Call function with M Flag = 1 specified.

Function: Calls a function in which M flag = 1

Format: #pragma M1FUNCTIONDfunction_name

Description: M and X flags are set as follows for a function declared in #pragma M1FUNCTION:

1.Calling function:
Sets the M flag to 1 and the X flag to 0, then calls the function

2. Called function:
Performs processing assuming that M flag is 1 and X flag is 0

Rule:

Specify the same settings for the M and X flags for both calling and called functions.

Example:

[C source program]

```
#pragma M1FUNCTION      func
char  func(char);

void main()
{
    func(1);
};


char func(char c)
{
    return c;
}
```

[Assembly language source program]

```
_main:
    .cline  6
    sem                    ←Sets M flag to 1
    lda.B   A,#01H
    pha
    jsrl    _func
          :
  (remainder omitted)
          :
```

Figure B.88   Example Use of #pragma M1FUNCT Declaration

B.7.7 The Other Extensions

NC77 includes the following extended function for embedding assembler description inline.

#pragma ASM, #pragma ENDASM

Inline assembling

Function : Specifies assembly code in C.

Syntax : #pragma ASM
assembly statements
#pragma ENDASM

Description : The line(s) between `#pragma ASM` and `#pragma ENDASM` are output without modifying anything to the generated assembly source file

Rules : Writing `#pragma ASM`, be sure to use it in combination with `#pragma ENDASM`. NC77 suspends processing if no `#pragma ENDASM` is found the corresponding `#pragma ASM`.

Example :

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

    for(i=0; i < 10;i++){
          func2();
    }

#pragma ASM
    CLI
LOOP1:
    LDX.W       #0000H
          :
      (omitted)
          :
    CLM
#pragma ENDASM

}
```

This area is output directly to an assembly language file.

Figure B.89 Example of #pragma ASM(ENDASM)

Note : It is this assembly language program written between `#pragma ASM` and `#pragma ENDASM` that is processed by the C preprocessor.

#pragma PAGE

Output .PAGE

Function : Declares new-page position in the assembler-generated list file.

Syntax : #pragma PAGE

Description : Putting the line `#pragma PAGE` in C source code, the .PAGE pseudo-instruction is output at the corresponding line in the compiler-generated assembly source. This instruction causes page ejection asesmbler-output assembly list file.

Rules :
1. You cannot specify the character string specified in the header of the assembler pseudo-instruction .PAGE.
2. You cannot write a #pragma PAGE in an auto variable declaration.

Example :

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

    for(i=0; i < 10;i++){
          func2();
    }

#pragma PAGE

    i++;

}
```

Figure B.90 Example of #pragma PAGE

B.8 assembler Macro Function

B.8.1 Outline of Assembler Macro Function

NC77 allows part of assembler commands to be written as C-language functions. Because specific assembler commands can be written directly in a C-language program, you can easily tune up the program.

B.8.2 Description Example of Assembler Macro Function

Assembler macro functions can be written in a C-language program in the same format as C-language functions, as shown below.

```
#include <asmmacro.h>        /* Includes the assembler macro function definition file  */
char    dest[20];
char    src[20];

func()
{
        mvn(dest,src,20);/* asm Macro Function(mvn command) */
}
```

Figure B.91 Description Example of Assembler Macro Function

B.8.3 Commands that Can be Written by Assembler Macro Function

The following shows the assembler commands that can be written using assembler macro functions and their functionality and format as assembler macro functions.

asl

Function : Returns the result after arithmetically shifting it to left as 1 time.

Syntax : #include <asmmacro.h>

```
char   asl_b(char val);          /* When calculated in 8 bits */
short  asl_w(short  val);        /* When calculated in 16 bits */
```

asr

Function : Returns the result after arithmetically shifting it to right as 1 time.
(only 775x)

Syntax : #include <asmmacro.h>

```
char   asr_b(char val);          /* When calculated in 8 bits */
short  asr_w(short  val);        /* When calculated in 16 bits */
```

div

Function : Returns the result where the dividend is a 4-byte value, the divisor is a 2-byte value.

Syntax : #include <asmmacro.h>

```
unsigned short    div_w(unsigned long val1, unsigned short  val2);
```

divs

Function : Returns the result where the dividend is a 4-byte value, the divisor is a 2-byte value. (only 775x)

Syntax : #include <asmmacro.h>

```
signed short       divs_w(signed long val1, signed short  val2);
```

lsr

Function : The value of val is returned after logically shifting it to right as 1 time.

Syntax : #include <asmmacro.h>

```
char   lsr_b(char val);           /* When calculated in 8 bits */
short  lsr_w(short  val);         /* When calculated in 16 bits */
```

mvn

Function : Strings are transferred by MVN instruction from the source address indicated by src to the destination address indicated by dest as many times as indicated by num. There is no return value.

Syntax : #include <asmmacro.h>

```
void   mvn(char _near * _near dest, char near * near src, short num);
```

mvp

Function : Strings are transferred by MVP instruction from the source address indicated by src to the destination address indicated by dest as many times as indicated by num. There is no return value.

Syntax : #include <asmmacro.h>

```
void   mvp(char _near * _near dest, char near * near src, short num);
```

rol

Function : The value of val is returned after rotating it left by 1 bit including the C flag.

Syntax : #include <asmmacro.h>

```
char   rol_b(char val);     /* When calculated in 8 bits */
short  rol_w(short val);    /* When calculated in 16 bits*/
```

ror

Function : The value of val is returned after rotating it right by 1 bit including the C flag.

Syntax : #include <asmmacro.h>

```
char   ror_b(char val);     /* When calculated in 8 bits */
short  ror_w(short val);    /* When calculated in 16 bits*/
```

Appendix C
Overview of C Language Specifications

In addition to the standard versions of C available on the market, C language specifications include extended functions for embedded system.

C.1 Performance Specifications

C.1.1 Overview of Standard Specifications

NC77 is a cross C compiler targeting the 7700 family. In terms of language specifications, it is virtually identical to the standard full-set C language, but also has specifications to the hardware in the 7700 family and extended functions for embedded system.

- Extended functions for embedded system(near/far modifiers, and asm function, etc.)
- Floating point library and host machine-dependent functions are contained in the standard library.

C.1.2 Introduction to NC77 Performance

This section provides an overview of NC77 performance.

a. Test Environment

Table C.1 shows the standard EWS environment assumed when testing performance. TableC.2 shows the standard PC environment.

TableC.1 Standard EWS Environment

Item	Type of EWS	UNIX Version
EWS environment	SPARCstation	SunOS V.4.1.3 JLE1.1.3
		Nihongo Solaris 2.5
	HP 9000/700 Series	HP-UX V.10.20
Available swap area	50MB min.	

TableC.2 Standard PC Environment

Item	Type of PC	DOS Version
PC environment	IBM PC/AT or compatible	Windows 95 Windows 98 Windows NT 4.0
Type of CPU	Intel Pentium	
Memory	32MB min.	

b. C Source File Coding Specifications

Table C.3 shows the specifications for coding NC77 C source files. Note that estimates are provided for items for which actual measurements could not be achieved.

TableC.3 Specifications for Coding C Source Files

Item	Specification
Number of characters per line of source file	512 bytes (characters) including the new line code
Number of lines in source file	65535 max.

c. NC77 Specifications

Table C.4 to C.5 lists the NC77 specifications. Note that estimates are provided for items for which actual measurements could not be achieved.

Table C.4 NC77 Specifications

Item	Specification
Maximum number of files that can be specified in NC77	Depends on amount of available memory
Maximum length of filename	Depends on operating system
Maximum number of macros that can be specified in nc77command line option -D	Depends on amount of available memory
Maximum number of directories that can be specified in nc77 command line option -I	8max
Maximum number of parameters that can be specified in nc77 command line option -rasm77	Depends on amount of available memory
Maximum number of parameters that can be specified in nc77 command line option -link77	Depends on amount of available memory
Maximum nesting levels of compound statements,iteration control structures, and selection control structures	Depends on amount of available memory
Maximum nesting levels in conditional compiling	
Number of pointers modifying declared basic types, arrays, and function declarators	Depends on amount of available memory
Number of function definitions	Depends on amount of available memory
Number of identifiers with block scope in one block	Depends on amount of available memory
Maximum number of macro identifiers that can be simultaneously defined in one source file	Depends on amount of available memory
Maximum number of macro name replacements	Depends on amount of available memory
Number of logical source lines in input program	Depends on amount of available memory
Maximum number of levels of nesting #include files	Depends on amount of available memory
Maximum number of case names in one switch statement (with no nesting of switch statement)	8max
Total number of operators and operands that can be defined in #if and #elif	Depends on amount of available memory
Size of stack frame that can be secured per function(in bytes)	Depends on amount of available memory
Number of variables that can be defined in #pragma ADDRESS	255max
Maximum number of levels of nesting parentheses	Depends on amount of available memory
Number of initial values that can be defined when defining variables with initialization expressions	Depends on amount of available memory Depends on amount of available memory
Maximum number of levels of nesting modifier declarators	
Maximum number of levels of nesting declarator parentheses	Depends on stack size of YACC
Maximum number of levels of nesting operator parentheses	Depends on stack size of YACC
Maximum number of valid characters per internal identifier or macro name	Depends on stack size of YACC
Maximum number of valid characters per external identifier	Depends on amount of available memory
Maximum number of external identifiers per source file	Depends on amount of available memory
Maximum number of identifiers with block scope per block	Depends on amount of available memory
Maximum number of macros per source file	Depends on amount of available memory
Maximum number of parameters per function call and per function	Depends on amount of available memory
Maximum number of parameters or macro call parameters per macro	Depends on amount of available memory
Maximum number of characters in character string literals after concatenation	31max
Maximum size (in bytes) of object	Depends on amount of available memory
Maximum number of members per structure/union	Depends on amount of available memory
Maximum number of enumerator constants per enumerator	Depends on amount of available memory
Maximum number of levels of nesting of structures or unions per struct declaration list	Depends on amount of available memory
Maximum number of characters per character string	Depends on amount of available memory
Maximum number of lines per file	Depends on operating system
	Depends on amount of available memory

C.2 Standard Language Specifications

The chapter discusses the NC77 language specifications with the standard language specifications.

C.2.1 Syntax

This section describes the syntactical token elements. In NC77, the following are processed as tokens:

- Key words
- Identifiers
- Constants
- Character literals
- Operators
- Punctuators
- Comment

a. Key Words

NC77 interprets the followings as key words.

Table C.5 Key Words List

_asm	default	if	struct
_far	do	int	switch
_near	double	long	typedef
asm	else	near	union
auto	enum	register	unsigned
break	extern	return	void
case	far	short	volatile
char	float	signed	while
const	for	sizeof	inline
continue	goto	static	

b. Identifiers

Identifiers consist of the following elements:

- The 1st character is a letter or the underscore (A to Z, a to z, or __)
- The 2nd and subsequent characters are alphanumerics or the underscore (A to Z, a to z, 0 to 9, or __)

Identifiers can consist of up to 31 characters. However, you cannot specify Japanese characters in identifiers.

c. Constants

Constants consists of the followings.

- ● Integer constants
- ● Floating point constants
- ● Character constants

(1)Integer constants

In addition to decimals, you can also specify octal and hexadecimal integer constants. Table C.6 shows the format of each base (decimal, octal, and hexadecimal).

Table C.6   Specifying Integer Constants

Base	Notation	Structure	Example
Decimal	None	0123456789	15
Octal	Start with 0 (zero)	01234567	017
Hexadecimal	Start with 0X or 0x	0123456789ABCDEF 0123456789abcdef	0XF or 0xf

Determine the type of the integer constant in the following order according to the value.

- ●Octal and hexadecimal : signed int ⇒unsigned int ⇒signed long ⇒unsigned long
- ●Decimal : signed int ⇒signed long ⇒unsigned long

Adding the suffix U or u, or L or l, results in the integer constant being processed as follows:

[1]Unsigned constants

Specify unsigned constants by appending the letter U or u after the value. The type is determined from the value in the following order:

- ●unsigned int ⇒unsigned long

[2]long-type constants

Specify long-type constants by appending the letter L or l. The type is determined from the value in the following order:

- ●signed long ⇒unsigned long

(2)Floating point constants

If nothing is appended to the value, floating point constants are handled as double types. To have them processed as float types, append the letter F or f after the value. If you append L or l, they are treated as long double types.

(3)Character constants

Character constants are normally written in single quote marks, as in 'character'. You can also include the following extended notation (escape sequences and trigraph sequences). Hexadecimal values are indicated by preceding the value with \x. Octal values are indicated by preceding the value with \.

Table C.7 Extended Notation List

Notation	Escape sequence	Notation	Trigraph sequence
\'	single quote	\constant	octal
\"	quotation mark	\xconstant	hexadecimal
\\	backslash	??(	express "[" character
\?	question mark	??/	express "\" character
\a	bell	??)	express "]" character
\b	backspace	??'	express "^" character
\f	form feed	??<	express "{" character
\n	line feed	??!	express "|" character
\r	return	??>	express "}" character
\t	horizontal tab	??-	express "~" character
\v	vertical tab	??=	express "#" character

d. Character Literals

Character literals are written in double quote marks, as in "character string". The extended notation shown in Table C.7 for character constants can also be used for character literals.

e. Operators

NC77 can interpret the operators shown in Table C.9.

Table C.8 Operators List

monadic operator	++	logical operator	&&
	--		||
	-		!
binary operator	+	conditional operator	?
	-	comma operator	,
	*	address operator	&
	/	pointer operator	*
	%	bitwise operator	<<
assignment operators	=		>>
	+=		&
	-=		|
	*=		^
	/=		~
	%=		&=
relational operators	>		|=
	<		^=
	>=		<<=
	<=		>>=
	==	sizeof operator	sizeof
	!=		

f. Punctuators

NC77 interprets the followings as punctuators.

- {
- }
- :
- ;
- ,

g. Comment

Comments are enclosed between /∗ and ∗/ . They cannot be nested.

C.2.2 Type

a. Data Type

NC77 supports the following data type.

- character type
- integral type
- structure
- union
- enumerator type
- void
- floating type

b. Qualified Type

NC77 interprets the following as qualified type.

- const
- volatile
- near
- far

c. Data Type and Size

Table C.9 shows the size corresponding to data type.

Table C.9   Data Type and Bit Size

Type	Existence of sign	Bit size	Range of values
char	No	8	0↔255
unsigned char			
signed char	Yes	8	-128↔127
int	Yes	16	-32768↔32767
short			
signed int			
signed short			
unsigned int	No	16	0↔65535
unsigned short			
long	Yes	32	-2147483648↔2147483647
signed long			
unsigned long	No	32	0↔4294967295
float	Yes	32	1.17549435e-38F↔3.40282347e+38F
double	Yes	64	2.2250738585072014e-308↔ 1.7976931348623157e+308
long double			
near pointer	No	16	0↔0xFFFF
far pointer	No	32	0↔0xFFFFFFFF

- If a char type is specified with no sign, it is processed as an unsigned char type.
- If an int or short type is specified with no sign, it is processed as a signed int or signed short type.
- If a long type is specified with no sign, it is processed as a sign long type.
- If the bit field members of a structure are specified with no sign,they are processed as unsigned.

C.2.3 Expressions

Tables C.10 and Table C.11 show the relationship between types of expressions and their elements.

Table C.10   Types of Expressions and Their Elements (1/2)

Type of expression	Elements of expression
Primary expression	identifier
	constant
	character literal
	(expression)
	primary expression
Postpositional expression	Postpositional expression [expression]
	Postpositional expression (list of parameters, ...)
	Postpositional expression. identifier
	Postpositional expression –>identifier
	Postpositional expression ++
	Postpositional expression ––
	Postpositional expression

Table C.11 Types of Expressions and Their Elements (2/2)

<table>
<tr><th>Type of expression</th><th>Elements of expression</th></tr>
<tr><td rowspan="6">Monadic expression</td><td>++ monadic expression</td></tr>
<tr><td>−− monadic expression</td></tr>
<tr><td>monadic operator cast expression</td></tr>
<tr><td>sizeof monadic expression</td></tr>
<tr><td>sizeof (type name)</td></tr>
<tr><td>Monadic expression</td></tr>
<tr><td rowspan="2">Cast expression</td><td>(type name) cast expression</td></tr>
<tr><td>cast expression</td></tr>
<tr><td rowspan="3">Expression</td><td>expression * expression</td></tr>
<tr><td>expression / expression</td></tr>
<tr><td>expression % expression</td></tr>
<tr><td rowspan="2">Additional and subtraction expressions</td><td>expression + expression</td></tr>
<tr><td>expression – expression</td></tr>
<tr><td rowspan="2">Bitwise shift expression</td><td>expression << expression</td></tr>
<tr><td>expression >> expression</td></tr>
<tr><td rowspan="5">Relational expressions</td><td>expression</td></tr>
<tr><td>expression < expression</td></tr>
<tr><td>expression > expression</td></tr>
<tr><td>expression <= expression</td></tr>
<tr><td>expression >= expression</td></tr>
<tr><td rowspan="2">Equivalence expression</td><td>expression == expression</td></tr>
<tr><td>expression != expression</td></tr>
<tr><td>Bitwise AND</td><td>expression & expression</td></tr>
<tr><td>Bitwise XOR</td><td>expression ^ expression</td></tr>
<tr><td>Bitwise OR</td><td>expression | expression</td></tr>
<tr><td>Logical AND</td><td>expression && expression</td></tr>
<tr><td>Logical OR</td><td>expression || expression</td></tr>
<tr><td>Conditional expression</td><td>expression ? expression: expression</td></tr>
<tr><td rowspan="11">Assign expression</td><td>monadic expression += expression</td></tr>
<tr><td>monadic expression −= expression</td></tr>
<tr><td>monadic expression *= expression</td></tr>
<tr><td>monadic expression /= expression</td></tr>
<tr><td>monadic expression %= expression</td></tr>
<tr><td>monadic expression <<= expression</td></tr>
<tr><td>monadic expression >>= expression</td></tr>
<tr><td>monadic expression &= expression</td></tr>
<tr><td>monadic expression |= expression</td></tr>
<tr><td>monadic expression ^= expression</td></tr>
<tr><td>assignment expression</td></tr>
<tr><td>Comma operator</td><td>expression, monadic expression</td></tr>
</table>

C.2.4 Declaration

There are two types of declaration:

- ●Variable Declaration
- ●Function Declaration

a. Variable Declaration

Use the format shown in Figure C.1 to declare variables.

```
storage class specifierΔtype declaratorΔdeclaration specifierΔinitialization_expression;
```

Figure C.1 Declaration Format of Variable

(1)Storage-class Specifiers

NC77 supports the following storage-class specifiers.

●extern	●auto	●typedef
●static	●register	

(2)Type Declarator

NC77 supports the type declarators.

●char	●long	●unsigned	●union
●int	●float	●signed	●enum
●short	●double	●struct	

(3)Declaration Specifier

Use the format of declaration specifier shown in Figure C.2 in NC77.

Declarator : Pointer$_{opt}$ declarator2

Declarator2 : identifier(declarator)

declarator2[constant expression$_{opt}$]

declarator2(list of dummy arguments$_{opt}$)

* Only the first array can be omitted from constant expressions showing the number of arrays.

* opt indicates optional items.

Figure C.2 Format of Declaration Specifier

(4)Initialization expressions

NC77 allows the initial values shown in Figure C.3 in initialization expressions.

integral types	:	constant
integral types array	:	constant, constant ....
character types	:	constant
character types array	:	character literal, constant ....
pointer types	:	character literal
pointer array	:	character literal, character literal ....

Figure C.3 Initial Values Specifiable in Initialization Expressions

b. Function Declaration

Use the format shown in Figure C.4 to declare functions.

● function declaration (definition)

storage-class specifierΔtype declaratorΔdeclaration specifierΔmain program

● function declaration (prototype declaration)

storage-class specifierΔtype declaratorΔdeclaration specifier;

Figure C.4 Declaration Format of Function

(1)Storage-class Specifier

NC77 supports the following storage-class specifier.

- extern
- static

(2)Type Declarators

NC77 supports the following type declarators.

- char
- int
- short
- long
- float
- double
- signed
- unsigned
- struct
- union
- enum

(3)Declaration Specifier

Use the format of declaration specifier shown in Figure C.5 in NC77.

Declarator	:	Pointer$_{opt}$ declarator2
Declarator2	:	identifier(list of dummy argument$_{opt}$)
		(declarator)
		declarator[constant expression$_{opt}$]
		declarator(list of dummy argument$_{opt}$)

* Only the first array can be omitted from constant expressions showing the number of arrays.
* opt indicates optional items.
* The list of dummy arguments is replaced by a list of type declarators in a prototype declaration.

Figure C.5 Format of Declaration Specifier

(4)Body of the Program

Use the format of body of the program shown in Figure C.6

List of Variable Declarator$_{opt}$ Compound Statement

*There is no body of the program in a prototype declaration, which ends with a semicolon.
*opt indicates optional items.

Figure C.6 Format of Body of the Program

C.2.5 Statement

NC77 supports the following.

- Labelled Statement
- Compound Statement
- Expression / Null Statement
- Selection Statement
- Iteration Statement
- Jump Statement
- Assembly Language Statement

a. Labelled Statement

Use the format of labelled statement shown in Figure C.7

Identifier	:	statement
case constant	:	statement
default	:	statement

Figure C.7 Format of Labelled Statement

b. Compound Statement

Use the format of compound statement shown in Figure C.8.

{ list of declarations$_{opt}$list of statements$_{opt}$ }

* opt indicates optional items.

Figure C.8 Format of Compound Statement

c. Expression / Null Statement

Use the format of expression and null statement shown in Figure C.9.

expression:
expression;
null statement:
;

Figure C.9 Format of Expression and Null Statement

d. Selection Statement

Use the format of selection statement shown in Figure C.10.

if(expression)statement
if(expression)statement else statement
switch(expression)statement

Figure C.10 Format of Selection Statement

e. Iteration Statement

Use the format of iteration statement shown in Figure C.11.

while(expression)statement
do statement while (expression);
for(expression$_{opt}$;expression$_{opt}$;expression$_{opt}$)statement;

* opt indicates optional items.

Figure C.11 Format of Iteration Statement

f. Jump statement

Use the format of jump statement shown in Figure C.12.

```
goto identifier;
continue;
break;
return expression_opt;

*opt indicates optional items.
```

Figure C.12 Format of Jump Statement

g. Assembly Language Statement

Use the format of assembly language shown in Figure C.13.

```
asm( "Literals" );

literals    :    assembly language statement
```

Figure C.13 Format of Assembly Language Statement

C.3 Preprocess Commands

Preprocess commands start with the pound sign (#) and are processed by the cpp77 preprocessor. This chapter provides the specifications of the preprocess commands.

C.3.1 List of Preprocess Commands Available

Table C.12 lists the preprocess commands available in NC77.

Table C.12 List of Preprocess Commands

Command	Function
#define	Defines macros.
#undef	Undefines macros.
#include	Takes in the specified file.
#error	Outputs messages to the standard output device and terminates processing.
#line	Specifies file's line numbers.
#assert	Outputs a warning when a constant expression is false.
#pragma	Instructs processing for NC77's extended function.
#if	Performs conditional compilation.
#ifdef	Performs conditional compilation.
#ifndef	Performs conditional compilation.
#elif	Performs conditional compilation.
#else	Performs conditional compilation.
#endif	Performs conditional compilation.

C.3.2 Preprocess Commands Reference

The NC77 preprocess commands are described in more detail below. They are listed in the order shown in Table C.12.

#define

[Function] Defines macros.

[Format] [1]#define∆ indentifier∆ lexical string opt
[2]#define∆ identifier (identifier list opt)∆ lexical string opt

[Description] [1]Defines an identifier as macro.
[2]Defines an identifier as macro. In this format, do not insert any space or tab between the first identifier and the left parenthesis '('.

- The identifier in the following code is replaced by blanks.

```
#define SYMBOL
```

- When a macro is used to define a function, you can insert a backslash so that the code can span two or more lines.
- The following four identifiers are reserved words for the compiler.

```
__FILE__ ............................. Name of source file
__LINE__ ............................ Current source file line No.
__DATE__ ........................... Date compiled (mm dd yyyy)
__TIME__ ............................ Time compiled (hh:mm:ss)
```

The following are predefined macros in NC77.

```
NC77
MELPS
MELPS7700
```

- You can use the token string operator '#' and token concatenated operator '##' with tokens, as shown below.

```
#define  debug(s,t)  printf("x"#s" = %d x"#t" = %d",x ## s,x ## t)

When parameters are specified for this macro debug (s, t) as debug (1, 2), they
are interpreted as follows:

#define  debug(s,t)  printf("x1 = %d x2 = %d", x1,x2)
```

#define

● Macro definitions can be nested (to a maximum of 20 levels) as shown below.

```
#define XYZ1        100
#define XYZ2         XYZ1
         :
(abbreviated)
         :
#define XYZ20       XYZ19
```

#undef

[Function] Nullifies an identifier that is defined as macro.

[Format] #undef∆ identifier

[Description] ● Nullifies an identifier that is defined as macro.

● The following four identifiers are compiler reserved words. Because these identifiers must be permanently valid, do not undefine them with #undef.

```
__FILE__............................. Name of source file
__LINE__ ............................ Current source file line No.
__DATE__............................ Date compiled (mm dd yyyy)
__TIME__............................. Time compiled (hh:mm:ss)
```

#include

[Function] Takes in the specified file.

[Format] [1]#include∆ <file name>
[2]#include∆ "file name"
[3]#include∆ identifier

[Description] [1]Takes in <file name> from the directory specified by nc77's command line option -I. Searches <file name> from the directory specified by environment variable "INC77" if it's not found.

[2]Takes in "file name" from the current directory.Searches "file name" from the following directory in sequence if it's not found.

1.The directory specified by nc77's startup option -I.
2.The directory specified by environment variable "INC77"

[3]If the macro-expanded identifier is <file name> or "file name" this command takes in that file from the directory according to rules of search [1]or [2].

- The maximum number of levels of nesting is 8.
- An include error results if the specified file does not exist.

#error

[Function] Suspends compilation and outputs the message to the standard output device.

[Format] #error∆character string

[Description]
- Suspends compilation.
- lexical string is found, this command outputs that character string to the standard output device.

#line

[Function] Changes the line number in the file.

[Format] #line∆ integer∆ "file name"

[Description]
- Specify the line number in the file and the filename.
- You can change the name of the source file and the line No.
- The maximum line No. is 9999. If the line No. is greater than 9999, no source line information is output as debugging information.

#assert

[Function] Issues a warning if a constant expression results in zero (0).

[Format] #assert∆ constant expression

[Description]
- Issues a warning if a constant expression results in zero (0). Compile is continued, however.

```
[Warning(cpp77.82):x.c, line xx]assertion warning
```

#pragma

[Function] Instructs the system to process NC77's extended functions.

[Format]

1.#pragma SECTIONΔ predetermined section nameΔ altered section name
2.#pragma ROMΔ variable name
3.#pragma STRUCTΔ tag name of structureΔ unpack
3.#pragma STRUCTΔ tag name of structureΔ arrange
4.#pragma INTERRUPTΔ interrupt handling function name
4.#pragma INTFΔ interrupt handling function name
5.#pragma ADDRESSΔ variable nameΔ absolute address
5.#pragma EQUΔ variable name = absolute address
6.#pragma PARAMETERΔ assembler function name(register name, register name, ..)
7.#pragma INTHANDLERΔ interrupt handler function name
7.#pragma HANDLERΔ interrupt handler function name
8.#pragma ALMHANDLERΔ alarm handler function name
9.#pragma CYCHANDLERΔ cyclic handler function name
10.#pragma TASKΔ task start function name
11.#pragma LOADDTΔ function name
12.#pragma M1FUNCTIONΔ function name
13.#pragma ASM
13.#pragma ENDASM
14.#pragma PAGE

[Description]

1.Facility to alter the section base name
2.Facility to arrange in the rom section
3.Facility to control the array of structures
4.Facility to write interrupt functions
5.Facility to specify absolute addresses for input/output variables
6.Facility to declare assembler functions passed via register
7.Facility to write interrupt handler functions
8.Facility to write alarm handler functions
9.Facility to write cyclic handler functions
10.Facility to write task start functions
11.Facility to specify DT register load functions
12.M flag setting function
13.Facility to describe inline assembler
14.Facility to output .PAGE

● You can only specify the above 14 processing functions with #pragma. If you specify a character string or identifier other than the above after #pragma, it will be ignored.
● Always use uppercase to specify the process (SECTION, INTERRUPT, etc.).
● By default, no warning is output if you specify an unsupported #pragma function. Warnings are only output if you specify the nc77 command line option -Wunknown_pragma (-WUP).

#if - #elif - #else - #endif

[Function] Performs conditional compilation.(Examines the expression true or false.)

[Format]

```
#if∆ constant expression
        :
#elif∆ constant expression
        :
#else
        :
#endif
```

[Description]

- If the value of the constant is true (not 0), the commands #if and #elif process the program that follows.
- #elif is used in a pair with #if, #ifdef, or #ifndef.
- #else is used in a pair with #if.Do not specify any tokens between #else and the line feed.You can, however, insert a comment.
- #endif indicates the end of the range controlled by #if. Always be sure to enter #endif when using command #if.
- Combinations of #if-#elif-#else-#endif can be nested.There is no set limit to the number of levels of nesting (but it depends on the amount of available memory).
- You cannot use the sizeof operator, cast operator, or variables in a constant expression.

#ifdef - #elif - #else - #endif

[Function] Performs conditional compilation.(Examines the macro defined or not.)

[Format]

```
#ifdefΔ identifier
        :
#elifΔ constant expression
        :
#else
        :
#endif
```

[Description]

● If an identifier is defined, #ifdef processes the program that follows.You can also describe the following.

```
#ifΔ definedΔ identifier
#ifΔ definedΔ (identifier)
```

● #else is used in a pair with #ifdef.Do not specify any tokens between #else and the line feed.You can, however, insert a comment.

● #elif is used in a pair with #if, #ifdef, or #ifndef.

● #endif indicates the end of the range controlled by #ifdef. Always be sure to enter #endif when using command #ifdef.

● Combinations of #ifdef-#else-#endif can be nested.There is no set limit to the number of levels of nesting (but it depends on the amount of available memory).

● You cannot use the sizeof operator, cast operator, or variables in a constant expression.

#ifndef - #elif - #else - #endif

[Function] Performs conditional compilation.(Examines the macro defined or not.)

[Format]

```
#ifndefΔ identifier
        :
#elifΔ constant expression
        :
#else
        :
#endif
```

[Description]

- If an identifier is defined, #ifdef processes the program that follows.You can also describe the followings.

```
#ifΔ !definedΔ identifier
#ifΔ !definedΔ (identifier)
```

- #else is used in a pair with #ifndef.Do not specify any tokens between #else and the line feed.You can, however, insert a comment.
- #elif is used in a pair with #if, #ifndef, or #ifndef.
- #endif indicates the end of the range controlled by #ifndef. Always be sure to enter #endif when using command #ifndef.
- Combinations of #ifndef-#else-#endif can be nested.There is no set limit to the number of levels of nesting (but it depends on the amount of available memory).
- You cannot use the sizeof operator, cast operator, or variables in a constant expression.

C.3.3 Predefined Macros

The following macros are predefined in NC77:

- NC77
- MELPS
- MELPS7700

C.3.4 Usage of predefined Macros

The predefined macros are used to, for example, use preprocess commands to switch machine-dependent code in non-NC77 C programs.

```
#ifdef  NC77
#pragma ADDRESS       port0     2H
#pragma ADDRESS       port1     3H

#else
#pragma AD            portA = 0x5F
#pragma AD            portB = 0x60

#endif
```

Figure C.14 Usage Example of Predefined Macros

Appendix D
C Language Specification Rules

This appendix describes the internal structure and mapping of data processed by NC77, the extended rules for signs in operations, etc., and the rules for calling functions and the values returned by functions.

D.1 Internal Representation of Data

Table D.1 shows the number of bytes used by integral type data.

D.1.1 Integral Type

Table D.1 Data Size of Integral Type

Type	Existence of Sign	Bit Size	Range of Values
char	No	8	0↔255
unsigned char			
signed char	Yes	8	-128↔127
int	Yes	16	-32768↔32767
short			
signed int			
signed short			
unsigned int	No	16	0↔65535
unsigned short			
long	Yes	32	-2147483648↔2147483647
signed long			
unsigned long	No	32	0↔4294967295

(1)If a char type is specified with no sign, it is processed as an unsigned char type.
(2)If an int or short type is specified with no sign, it is processed as a signed int or signed short type.
(3)If a long type is specified with no sign, it is processed as a sign long type.

D.1.2 Floating Type

Table D.2 shows the number of bytes used by floating type data.

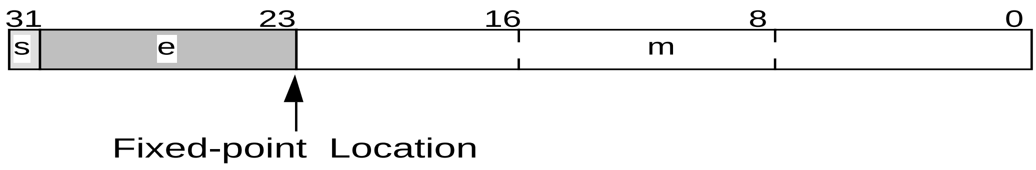
Table D.2 Data Size of Floating Type

Type	Existence of sign	Bit Size	Range of values
float	Yes	32	1.17549435e-38F↔3.40282347e+38F
double	Yes	64	2.2250738585072014e-308↔ 1.7976931348623157e+308
long double			

NC77's floating-point format conforms to the format of IEEE (Institute of Electrical and Electronics Engineers) standards. The following shows the single precision and double precision floating-point formats.

(1)Single-precision floating point data format

Figure D.1 shows the format for binary floating point (float) data.

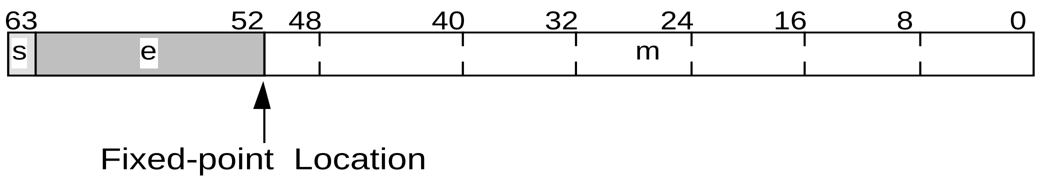


s:fixed-point part sign(1bit)
e:characteristic part(8bits)
m:fixed-point part(23bits)

Figure D.1   Single-precision floating point data format

(2)Double-precision floating point data format

Figure D.2 shows the format for binary floating point (double and long double) data.



s:fixed-point part sign(1bit)
e:characteristic part(11bits)
m:fixed-point part(52bits)

Figure D.2   Double-precision floating point data format

D.1.3   Enumerator Type

Enumerator types have the same internal representation as unsigned int types. Unless otherwise specified, integers 0, 1, 2, … are applied in the order in which the members appear.

Note that you can also use the nc77 command line option -fchar_enumerator (-fCE) to force enumerator types to have the same internal representation as unsigned char types.

D.1.4 Pointer Type

Table D.3 shows the number of bytes used by pointer type data.

Table D.3 Data Size of Pointer Types

Type	Existence of Sign	Bit Size	Range
near pointers	None	16	0-0xFFFF
far pointers	None	32	0-0xFFFFFFFF

Note that only the least significant 24 bits of the 32 bits of far pointers are valid.

D.1.5 Array Types

Array types are mapped contiguously to an area equal to the product of the size of the elements (in bytes) and the number of elements. They are mapped to memory in the order in which the elements appear. Figure D.3 is an example of mapping.

(Example)

```
char  c[5] = {0, 1, 2, 3, 4};
```

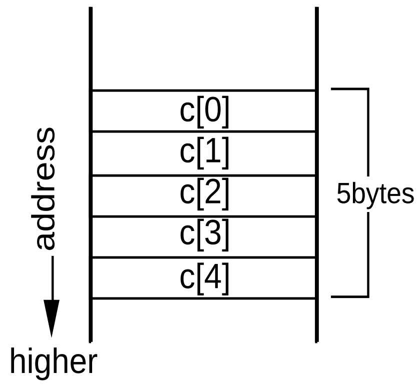


Figure D.3 Example of Placement of Array

D.1.6 Structure types

Structure types are mapped contiguously in the order of their member data. Figure D.4 is an example of mapping.

(Example)

```
struct TAG {
      char     c;
      int      i;
      } s;
```

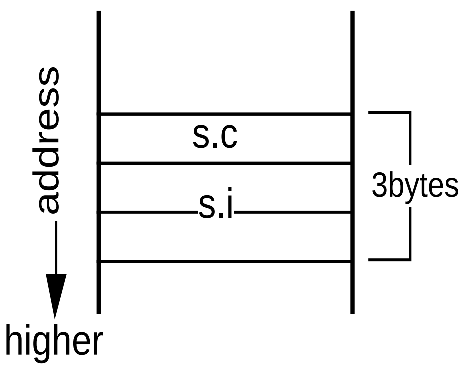


Figure D.4 Example of Placement of Structure(1/2)

Normally, there is no word alignment with structures. The members of structures are mapped contiguously. To use word alignment, use the #pragma STRUCT extended function. #pragma STRUCT adds a byte of padding if the total size of the members is odd. Figure D.5 is an example of mapping.

(Example)

```
#pragma STRUCT TAG unpack
struct TAG {
        char      c;
        int       i;
        } s;
```

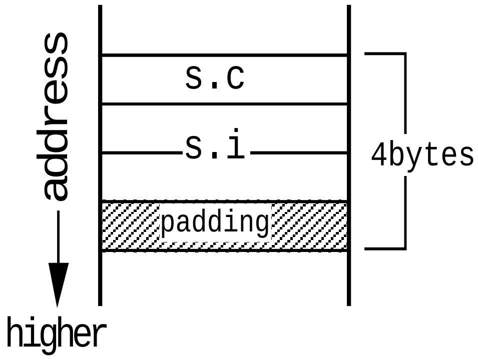


Figure D.5   Example of Placement of Structure(2/2)

D.1.7   Unions

Unions occupy an area equal to the maximum data size of their members. Figure D.6 is an example of mapping.

(Example)

```
union TAG {
      char    c;
      int     i;
      long    lo;
      } s;
```

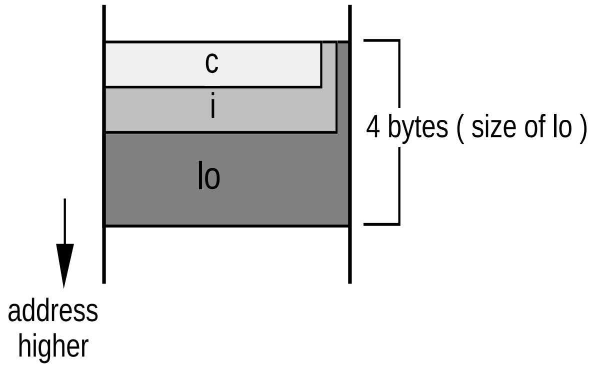


Figure D.6   Example of Placement of Union

D.1.8   Bitfield Types

Bitfield types are mapped from the least significant bit. Figure D.7 is an example of mapping.

(Example)

```
struct BTAG {
        char  b0 : 1;
        char  b1 : 1;
        char  b2 : 1;
        char  b3 : 1;
        char  b4 : 1;
        char  b5 : 1;
        char  b6 : 1;
        char  b7 : 1;
        } s;
```

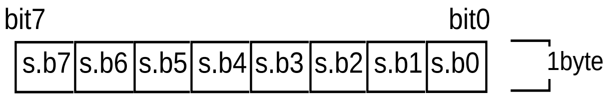


Figure D.7   Example of Placement of Bitfield(1/2)

If a bitfield member is of a different data type, it is mapped to the next address. Thus, members of the same data type are mapped contiguously from the lowest address to which that data type is mapped.

(Example)

```
struct BTAG {
        char  b0 : 1;
        char  b1 : 1;
        char  b2 : 1;
        char  b3 : 1;
        char  b4 : 1;
        int   b56: 2;
        char  b7 : 1;
} s;
```

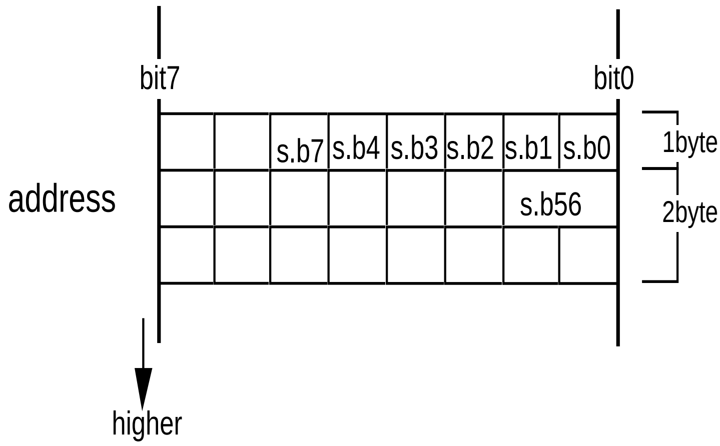


Figure D.8   Example of Placement of Bitfield(2/2)

If no sign is specified, the default bitfield member type is unsigned.

D.2 Sign Extension Rules

Under the ANSI and other standard C language specifications, char type data is sign extended to int type data for calculations, etc. This specification prevents the maximum value for char types being exceeded with unexpected results when performing the char-type calculation shown in Figure D.9.

```
func()
{
    char   c1, c2, c3;

    c1 = c2  *  2  /  c3;
}
```

Figure D.9 Example of C Program

To generate code that maximizes code efficiency and maximizes speed, NC77 does not, by default, extend char types to int types. The default can, however, be overridden using the nc77 compile driver command line option -fansi or -fextend_to_int (-fETI) to achieve the same sign extension as in standard C.

If you do not use the -fansi or -fextend_to_int (-fETI) option and your program assigns the result of a calculation to a char type, as in Figure D.9, make sure that the maximum or minimum[*1] value for a char type does not result in an overflow in the calculation.

D.3 Function Call Rules

D.3.1 Rules of Return Value

When returning a return value from a function, the system uses a register to return that value for the integer, pointer, and floating-point types. Table D.4 shows rules on calls regarding return values.

Table D.4 Return Value-related Calling Rules

Type of Return Value	Rules
char	A Register
int near pointer	A Register
float long far pointer	Least significant 16 bits returned by storing in A register. Most significant 16 bits returned by storing in B register.
double long double Structure Type Union Type	Immediately before the function call, save the far address for the area for storing the return value to the stack. Before execution returns from the called function, that function writes the return value to the area indicated by the far address saved to the stack.

*1. The ranges of values that can be expressed as char types in NC77 are as follows:
- * unsigned char type .... 0↔255
- * signed char type .... -128↔127

D.3.2 Rules on Argument Transfer

NC77 uses registers or stack to pass arguments to a function.

(1)Passing arguments via register

When the conditions below are met, the system uses the corresponding "Registers Used" listed in Table D.5 and D.6 to pass arguments.

- Function is prototype declared [*1] and the type of argument is known when calling the function.
- Variable argument "..." is not used in prototype declaration.
- For the type of the argument of a function, the Argument and Type of Argument in Table D.5 are matched.

Table D.5 Rules on Argument Transfer via Register

Argument	First Argument	Registers Used
First argument	char type	A register
	int type near pointer type	A register

(2)Passing arguments via stack

All arguments that do not satisfy the register transfer requirements are passed via stack. The table D.6 summarize the methods used to pass arguments.

Table D.6 Rules on Passing Arguments to Function

Type of Argument	First Argument	Second Argument
char type	A register	Stack
int type near pointer type	A register	Stack
Other types	Stack	Stack

*1. NC77 uses a via-register transfer only when entering prototype declaration (i.e., when writing a new format). Consequently, all arguments are passed via stack when description of K&R format is entered (description of old format).
Note also that if a description format where prototype declaration is entered for the function (new format) and a description of the K&R format (old format) coexist in given statement, the system may fail to pass arguments to the function correctly, for reasons of language specifications of the C language.
Therefore, we recommends using a prototype- declaring description format as the standard format to write the C language source files for NC77.

D.3.3 Rules for Converting Functions into Assembly Language Symbols

The function names in which functions are defined in a C language source file are used as the start labels of functions in an assembler source file.

The start label of a function in an assembler source file consists of the function name in the C language source file prefixed by _ (underscore) or ? (question).

The table below lists the character strings that are added to a function name and the conditions under which they are added.

Table D.7 Conditions Under Which Character Strings Are Added to Function

Added character string	Condition
? (question)	Functions where any one of arguments is passed via register
_ (underscore)	Functions that do not belong to the above*1

Shown in Figure D.10 is a sample program where a function has register arguments and where a function has its arguments passed via only a stack.

```
int func_proto( int , int , int);                    ←[1]

int func_proto(int i, int j, int k)                  [2]
{
        return i + j + k;
}

int func_no_proto( i, j, k)                          [3]
int i;
int j;
int k;
{
        return i + j + k;
}

void                                                 [4]
main(void)
{
        int sum;
        sum = func_proto(1,2,3);          ←[5]
        sum = func_no_proto(1,2,3);       ←[6]
}
```

[1]This is the prototype declaration of function func_proto.
[2]This is the body of function func_proto. (Prototype declaration is entered, so this is a new format.)
[3]This is the body of function func_no_proto. (This is a description in K&R format, that is, an old format.)
[4]This is the body of function main.
[5]This calls function func_proto.
[6]This calls function func_no_proto.

Figure D.10 Sample Program for Calling a Function (sample.c)

The compile result of the above sample program is shown in the next page. Figure D.11 shows the compile result of program part[2]that defines function func_proto.Figure D.12 shows the compile result of program part[3]that defines function func_no_proto.Figure D.13 shows the compile result of program part[4]that calls function func_proto and function func_no_proto.

*1. However, function names are not output for the functions that are specified by #pragma INTCALL.

```
;## # FUNCTION func_proto
;## # FRAME      AUTO (     i) size   2,      offset 1
;## # FRAME      ARG  (     j) size   2,      offset 8          ←[8]
;## # FRAME      ARG  (     k)        size   2,      offset 10  ←[7]
;## # REGISTER ARG     (     i) size   2,      REGISTER A       ←[9]
;## # ARG Size(4)  Auto Size(2)  Context Size(5)

        .source test.c
        .section        program_F
;## # C_SRC :       {
        .DT     __DT
        .DP     OFF
        .func   ?func_proto
        .pub    ?func_proto
?func_proto:                                        ←[10]
        phd
        pha     ; Register Argument
        tsa
        tad
        .cline  6
;## # C_SRC :          return i + j + k;
        lda     A,DP:8          ;  j
        clc
        adc     A,DP:1          ;  i
        clc
        adc     A,DP:10         ;  k
        plx
        pld
        rtl
        .endfunc        ?func_proto
```

[7]This passes the third argument k via stack.
[8]This passes the first argument i via register.
[9]This passes the second argument j via register.
[10]This is the start address of function func_proto.

Figure D.11 Compile Result of Sample Program (sample.c) (1/3)

In the compile result (1) of the sample program (sample.c) listed in Figure D.10, the first arguments is passed via a register since function func_proto is prototype declared. The second and third argument are passed via a stack since it is not subject to via-register transfer.

Furthermore, since the arguments of the function are passed via register, the symbol name of the function's start address is derived from "func_proto" described in the C language source file by prefixing it with ? (question), hence, "?func_proto."

```
;## # FUNCTION func_no_proto
;## # FRAME      ARG  (    i) size  2,      offset 6     [11]
;## # FRAME      ARG  (    j) size  2,      offset 8
;## # FRAME      ARG  (    k) size  2,      offset 10
;## # ARG Size(6)  Auto Size(0)  Context Size(5)

;## # C_SRC :     {
        .DT     __DT
        .DP     OFF
        .func   _func_no_proto
        .pub    _func_no_proto
_func_no_proto:                          ←[12]
        phd
        tsa
        tad
        .cline  14
;## # C_SRC :            return i + j + k;
        lda     A,DP:8          ;  j
        clc
        adc     A,DP:6          ;  i
        clc
        adc     A,DP:10         ;  k
        pld
        rtl
        .endfunc        _func_no_proto
```

[11]This passes all arguments via a stack.
[12]This is the start address of function func_no_proto.

Figure D.12 Compile Result of Sample Program (sample.c) (2/3)

In the compile result (2) of the sample program (sample.c) listed in Figure D.10, all arguments are passed via a stack since function func_no_proto is written in K&R format.

Furthermore, since the arguments of the function are not passed via register, the symbol name of the function's start address is derived from "func_no_proto" described in the C language source file by prefixing it with _ (underscore), hence, "_func_no_proto."

```
;## # FUNCTION main
;## # FRAME        AUTO (     sum)       size   2,        offset 1
;## # ARG Size(0)  Auto Size(2)  Context Size(5)

;## # C_SRC :       {
        .DT     __DT
        .DP     OFF
        .func   _main
        .pub    _main
_main:
        phd
        pha
        tsa
        tad
        .cline  21
;## # C_SRC :       sum = func_proto(1,2,3);
        pea     #0003H                              [11]
        pea     #0002H
        lda.W   A,#0001H
        jsrl    ?func_proto
        plx
        plx
        sta     A,DP:1          ;  sum
        .cline  22
;## # C_SRC :       sum = func_no_proto(1,2,3);
        pea     #0003H                              [12]
        pea     #0002H
        pea     #0001H
        jsrl    _func_no_proto
        tax
        tda
        tas
        txa
        sta     A,DP:1          ;  sum
        .cline  23
;## # C_SRC :       }
        plx
        pld
        rtl
        .endfunc        _main
```

Figure D.13 Compile Result of Sample Program (sample.c) (3/3)

In Figure D.13, part[11]calls func_proto and part[12]calls func_no_proto.

D.3.4 Interface between Functions

Figures D.16 to D.18 show the stack frame structuring and release processing for the program shown in Figure D.14. Figure D.15 shows the assembly language program that is produced when the program shown in Figure D.14 is compiled.

```
 int        func( int, int ,int)
void main(void)
{
        int             i = 0x1234;                  ←Argument to func
        int             j = 0x5678;                  ←Argument to func
        int             k = 0x9abc;                  ←Argument to func
        k = func( i, j ,k);
}

int func( int x,int  y,int  z )
{
         int  sum;
        sum = x + y + z ;

        return  sum;             ←Return value to main
}
```

Figure D.14   Example of C Language Sample Program

```
;## # FUNCTION main
;## # FRAME       AUTO (     k)       size   2,       offset 1
;## # FRAME       AUTO (     j) size   2,       offset 3
;## # FRAME       AUTO (     i) size   2,       offset 5
;## # ARG Size(0)  Auto Size(6)  Context Size(5)

        .source test.c
        .section        program_F
;## # C_SRC :        {
        .DT     __DT
        .DP     OFF
        .func   _main
        .pub    _main
_main:                                    ←[1]
        phd
        pha
        pha
        pha
        tsa
        tad                               ←[2]
        .cline  4
;## # C_SRC :           int         i = 0x1234;
        ldm.W #1234H,DP:5 ;  i
        .cline  5
;## # C_SRC :           int         j = 0x5678;
        ldm.W #5678H,DP:3 ;  j
        .cline  6
;## # C_SRC :           int         k = 0x9abc;
        ldm.W #9abcH,DP:1 ;  k
        .cline  7
;## # C_SRC :           k = func( i, j ,k);
        pei     #1      ;  k              ←[3]
        pei     #3      ;  j              ←[4]
        lda     A,DP:5        ;  i        ←[5]
        jsrl    ?func                     ←[6]
        plx
        plx                               ←[10]
        sta     A,DP:1        ;  k        ←[11]
        .cline  8
;## # C_SRC :        }
        plx
        plx
        plx
        pld
        rtl
        .endfunc        _main
```

Figure D.15 Assembly language sample program (1/2)

```
;## # FUNCTION func
;## # FRAME        AUTO (      x)        size   2,        offset 3
;## # FRAME        AUTO (    sum)        size   2,        offset 1
;## # FRAME        ARG   (      y)        size   2,        offset 10
;## # FRAME        ARG   (      z)        size   2,        offset 12
;## # REGISTER ARG      (      x)        size   2,        REGISTER A
;## # ARG Size(4)  Auto Size(4)  Context Size(5)

;## # C_SRC :       {
        .DT     __DT
        .DP     OFF
        .func   ?func
        .pub    ?func
?func:
        phd
        pha     ; Register Argument
        pha
        tsa
        tad                                   ←[7]
        .cline  13
;## # C_SRC :            sum = x + y + z ;
        lda     A,DP:10        ;  y
        clc
        adc     A,DP:3         ;  x
        clc
        adc     A,DP:12        ;  z
        sta     A,DP:1         ;  sum
        .cline  15
;## # C_SRC :            return  sum;
        lda     A,DP:1         ;  sum         ←[8]
        plx
        plx
        pld
        rtl                                   ←[9]
        .endfunc       ?func
```

Figure D.16 Assembly language sample program (2/2)

Figures D.16 to D.18 below show stack and register transitions in each processing in Figure D.15. Processing in[1]⇒[2](entry processing of function main) is shown in Figure D.16. Processing[3]⇒[4]⇒[5]⇒[6]⇒[7](processing to call function func and construct stack frames used in function func) is shown in Figure D.17.

Processing[8]⇒[9]⇒[10]⇒[11](processing to return from function func to function main) is shown in Figure D.18.

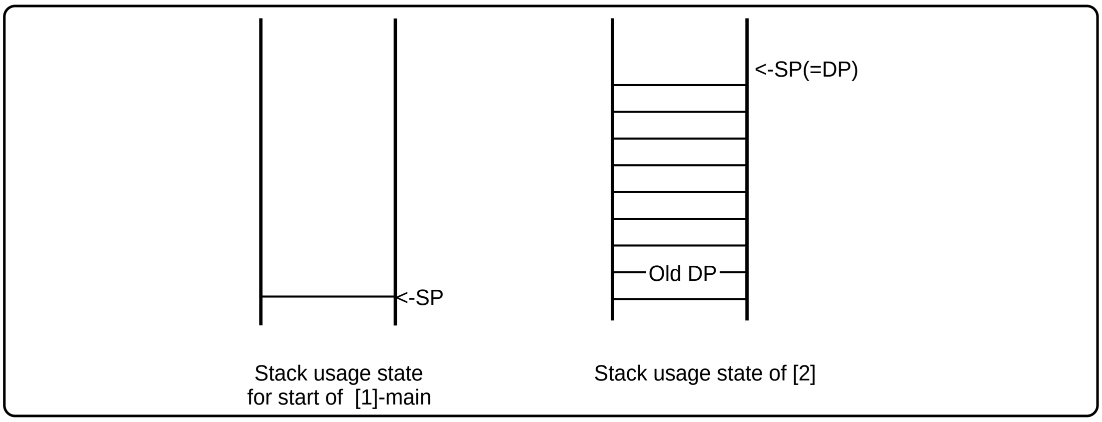


Figure D.17   Entry processing of function main

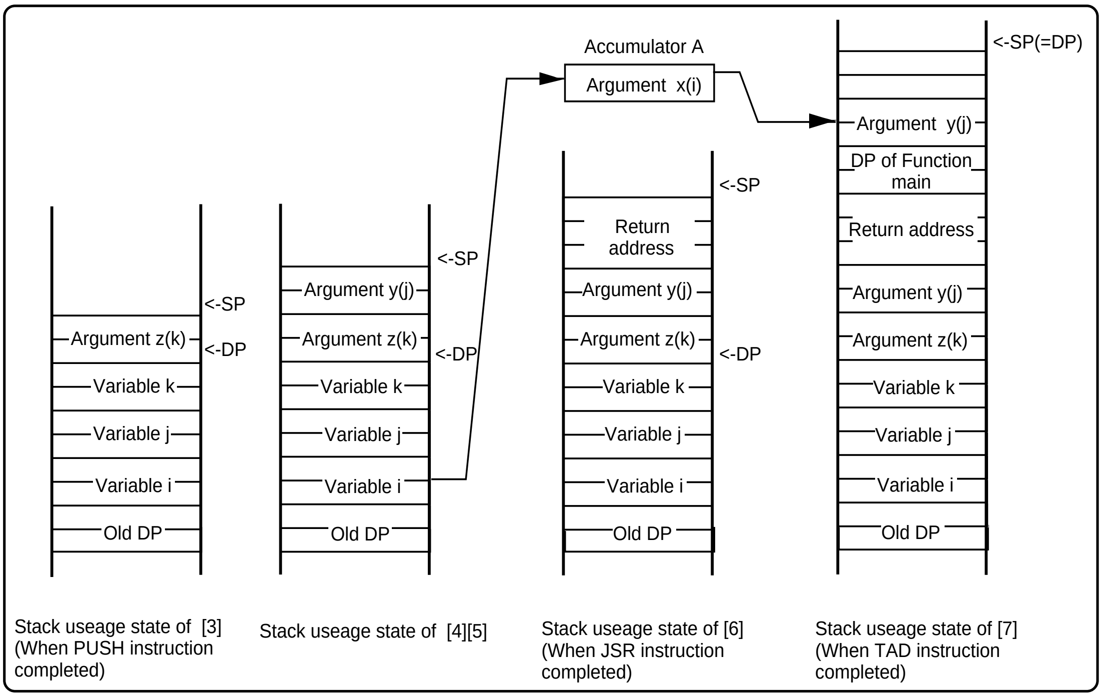


Figure D.18 Calling Function func and Entry Processing

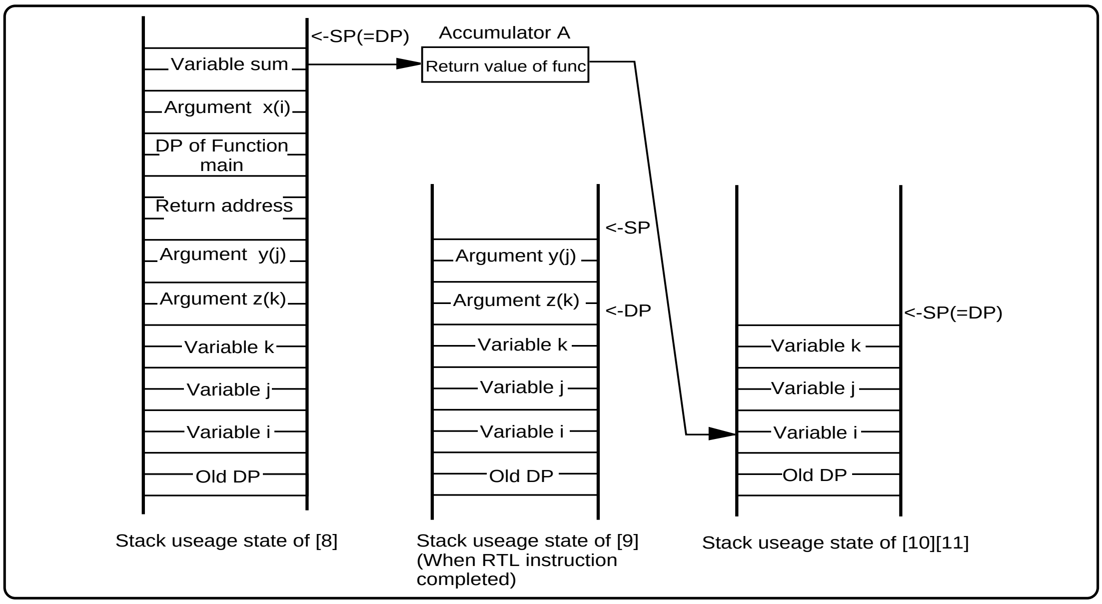


Figure D.19 Exit Processing of Function func

D.4 Securing auto Variable Area

Variables of storage class auto are placed in the stack of the 7700 family microcomputer. For a C language source file like the one shown in Figure D.20, if the areas where variables of storage class auto are valid do not overlap each other, the system allocates only one area which is then shared between multiple variables.

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

    for ( i=0 ;  i<=0 ;  i++ ) {
         process                                      scope of i
    }
         :
      (abbreviated)
         :
    for ( j=0xFF ;  j<=0  ;  j-- ) {
         process                                      scope of j
    }
         :
      (abbreviated)
         :
    for ( k=0 ;  k<=0  ;  k++ ){
         process                                      scope of k
    }
}
```

Figure D.20 Example of C Program

In this example, the effective ranges of three auto variables i, j, and k do not overlap, so that a two-byte area (offset from DPR) is shared . Figure D.21 shows an assembly language source file generated by compiling the program in Figure D.20.

```
 ## #FUNCTION func
;## # FRAME        AUTO (      k)         size   2,        offset 1       ←[1]
;## # FRAME        AUTO (      j) size   2,        offset 1              ←[2]
;## # FRAME        AUTO (      i) size   2,        offset 1              ←[3]
;## # ARG Size(0)  Auto Size(2)  Context Size(5)

        .source test.c
        .section        program_F
;## # C_SRC :       {
        .DT     __DT
        .DP     OFF
        .func   _func
        .pub    _func
_func:
        phd
        pha
        tsa
        tad
        .cline  7
;## # C_SRC :           for( i=0 ; i<=0 ; i++ ){
```

* As shown by [1],[2], and [3],the three auto variables share the DPR offset 1 area.

Figure D.21   Example of Assembly Language Source Program

Appendix E
Standard Library

E.1 Standard Header Files

When using the NC77 standard library, you must include the header file that defines that function.

This appendix details the functions and specifications of the standard NC77 header files.

E.1.1 Contents of Standard Header Files

NC77 includes the 15 standard header files shown in Table E.1.

Table E.1 List of Standard Header Files

Header File Name	Contents
assert.h	Outputs the program's diagnostic information.
ctype.h	Declares character determination function as macro.
errno.h	Defines an error number.
float.h	Defines various limit values concerning the internal representation of floating points.
limits.h	Defines various limit values concerning the internal processing of compiler.
locale.h	Defines/declares macros and functions that manipulate program localization.
math.h	Declares arithmetic/logic functions for internal processing.
setjmp.h	Defines the structures used in branch functions.
signal.h	Defines/declares necessary for processing asynchronous interrupts.
stdarg.h	Defines/declares the functions which have a variable number of real arguments.
stddef.h	Defines the macro names which are shared among standard include files.
stdio.h	Defines the FILE structure.
	Defines a stream name.
	Declares the prototype of input/output functions.
stdlib.h	Declares the prototypes of memory management and terminate functions.
string.h	Declares the prototypes of character string and memory handling functions.
time.h	Declares the functions necessary to indicate the current calendar time and defines the type.

E.1.2 Standard Header Files Reference

Following are detailed descriptions of the standard header files supplied with NC77. The header files are presented in alphabetical order.

The NC77 standard functions declared in the header files and the macros defining the limits of numerical expression of data types are described with the respective header files.

assert.h

[Function] Defines assert function.

ctype.h

[Function] Defines/declares string handling function.The following lists string handling functions.

Function	Contents
isalnum	Checks whether the character is an alphabet or numeral.
isalpha	Checks whether the character is an alphabet.
iscntrl	Checks whether the character is a control character.
isdigit	Checks whether the character is a numeral.
isgraph	Checks whether the character is printable (except a blank).
islower	Checks whether the character is a lower-case letter.
isprint	Checks whether the character is printable (including a blank).
ispunct	Checks whether the character is a punctuation character.
isspace	Checks whether the character is a blank, tab, or new line.
isupper	Checks whether the character is an upper-case letter.
isxdigit	Checks whether the character is a hexadecimal character.
tolower	Converts the character from an upper-case to a lower-case.
toupper	Converts the character from a lower-case to an upper-case.

errno.h

[Function] Defines error number.

float.h

[Function] Defines the limits of internal representation of floating point values. The following lists the macros that define the limits of floating point values.

In NC77, long double types are processed as double types. Therefore, the limits applying to double types also apply to long double types.

Macro name	Contents	Defined value
DBL_DIG	Maximum number of digits of double-type decimal precision	15
DBL_EPSILON	Minimum positive value where 1.0+DBL_EPSILON is found not to be 1.0	2.2204460492503131e-16
DBL_MANT_DIG	Maximum number of digits in the mantissa part when a double-type floating-point value is matched to the radix in its representation	53
DBL_MAX	Maximum value that a double-type variable can take on as value	1.7976931348623157e+308
DBL_MAX_10_EXP	Maximum value of the power of 10 that can be represented as a double-type floating-point numeric value	308
DBL_MAX_EXP	Maximum value of the power of the radix that can be represented as a double-type floating-point numeric value	1024
DBL_MIN	Minimum value that a double-type variable can take on as value	2.2250738585072014e-308
DBL_MIN_10_EXP	Minimum value of the power of 10 that can be represented as a double-type floating-point numeric value	-307
DBL_MIN_EXP	Minimum value of the power of the radix that can be represented as a double-type floating-point numeric value	-1021
FLT_DIG	Maximum number of digits of float-type decimal precision	6
FLT_EPSILON	Minimum positive value where 1.0+FLT_EPSILON is found not to be 1.0	1.19209290e-07F
FLT_MANT_DIG	Maximum number of digits in the mantissa part when a float-type floating-point value is matched to the radix in its representation	24
FLT_MAX	Maximum value that a float-type variable can take on as value	3.40282347e+38F
FLT_MAX_10_EXP	Maximum value of the power of 10 that can be represented as a float-type floating-point numeric value	38
FLT_MAX_EXP	Maximum value of the power of the radix that can be represented as a float-type floating-point numeric value	128
FLT_MIN	Minimum value that a float-type variable can take on as value	1.17549435e-38F
FLT_MIN_10_EXP	Minimum value of the power of 10 that can be represented as a float-type floating-point numeric value	-37
FLT_MIN_EXP	Maximum value of the power of the radix that can be represented as a float-type floating-point numeric value	-125
FLT_RADIX	Radix of exponent in floating-point representation	2
FLT_ROUNDS	Method of rounding off a floating-point number	1(Rounded to the nearest whole number)

limits.h

[Function] Defines the limitations applying to the internal processing of the compiler. The following lists the macros that define these limits.

Macro name	Contents	Defined value
CHAR_BIT	Number of char-type bits	8
CHAR_MAX	Maximum value that a char-type variable can take on as value	255
CHAR_MIN	Minimum value that a char-type variable can take on as value	0
INT_MAX	Maximum value that a int-type variable can take on as valueMaximum value that a int-type variable can take on as value	32767
INT_MIN	Minimum value that a int-type variable can take on as value	-32768
LONG_MAX	Maximum value that a long-type variable can take on as value	2147483647
LONG_MIN	Minimum value that a long-type variable can take on as value	-2147483648
MB_LEN_MAX	Maximum value of the number of multibyte character-type bytes	1
SCHAR_MAX	Maximum value that a signed char-type variable can take on as value	127
SCHAR_MIN	Minimum value that a signed char-type variable can take on as value	-128
SHRT_MAX	Maximum value that a short int-type variable can take on as value	32767
SHRT_MIN	Minimum value that a short int-type variable can take on as value	-32768
UCHAR_MAX	Maximum value that an unsigned char-type variable can take on as value	255
UINT_MAX	Maximum value that an unsigned int-type variable can take on as value	65535
ULONG_MAX	Maximum value that an unsigned long int-type variable can take on as value	4294967295
USHRT_MAX	Maximum value that an unsigned short int-type variable can take on as value	65535

locale.h

[Function] Defines/declares macros and functions that manipulate program localization.The following lists locale functions.

Function	Contents
localeconv	Initializes struct lconv.
setlocale	Sets and searches the locale information of a program.

math.h

[Function] Declares prototype of mathematical function.The following lists mathematical functions.

Function	Contents
acos	Calculates arc cosine.
asin	Calculates arc sine.
atan	Calculates arc tangent.
atan2	Calculates arc tangent.
ceil	Calculates an integer carry value.
cos	Calculates cosine.
cosh	Calculates hyperbolic cosine.
exp	Calculates exponential function.
fabs	Calculates the absolute value of a double-precision floating-point number.
floor	Calculates an integer borrow value.
fmod	Calculates the remainder.
frexp	Divides floating-point number into mantissa and exponent parts.
labs	Calculates the absolute value of a long-type integer.
ldexp	Calculates the power of a floating-point number.
log	Calculates natural logarithm.
log10	Calculates common logarithm.
modf	Calculates the division of a real number into the mantissa and exponent parts.
pow	Calculates the power of a number.
sin	Calculates sine.
sinh	Calculates hyperbolic sine.
sqrt	Calculates the square root of a numeric value.
tan	Calculates tangent.
tanh	Calculates hyperbolic tangent.

setjmp.h

[Function] Defines the structures used in branch functions.

Function	Contents
longjmp	Performs a global jump.
setjmp	Sets a stack environment for a global jump.

signal.h

[Function] Defines/declares necessary for processing asynchronous interrupts.

stdarg.h

[Function] Defines/declares the functions which have a variable number of real arguments.

stddef.h

[Function] Defines the macro names which are shared among standard include files.

stdio.h

[Function] Defines the FILE structure,stream name, and declares I/O function prototypes. Prototype declarations are made for the following functions.

Type	Function	Contents
Initialize	init	Initializes 7700 family input/outputs.
	clearerr	Initializes (clears) error status specifiers.
Input	fgetc	Inputs one character from the stream.
	getc	Inputs one character from the stream.
	getchar	Inputs one character from stdin.
	fgets	Inputs one line from the stream.
	gets	Inputs one line from stdin.
	fread	Inputs the specified items of data from the stream.
	scanf	Inputs characters with format from stdin.
	fscanf	Inputs characters with format from the stream.
	sscanf	Inputs data with format from a character string.
Output	fputc	Outputs one character to the stream.
	putc	Outputs one character to the stream.
	putchar	Outputs one character to stdout.
	fputs	Outputs one line to the stream.
	puts	Outputs one line to stdout.
	fwrite	Outputs the specified items of data to the stream.
	perror	Outputs an error message to stdout.
	printf	Outputs characters with format to stdout.
	fflush	Flushes the stream of an output buffer.
	fprintf	Outputs characters with format to the stream.
	sprintf	Writes text with format to a character string.
	vfprintf	Output to a stream with format.
	vprintf	Output to stdout with format.
	vsprintf	Output to a buffer with format.
Return	ungetc	Sends one character back to the input stream.
Determination	ferror	Checks input/output errors.
	feof	Checks EOF (End of File).

stdlib.h

[Function] Declares the prototypes of memory management and terminate functions.

Function	Contents
abort	Terminates the execution of the program.
abs	Calculates the absolute value of an integer.
atof	Converts a character string into a double-type floating- point number.
atoi	Converts a character string into an int-type integer.
atol	Converts a character string into a long-type integer.
bsearch	Performs binary search in an array.
calloc	Allocates a memory area and initializes it to zero (0).
div	Divides an int-type integer and calculates the remainder.
free	Frees the allocated memory area.
labs	Calculates the absolute value of a long-type integer.
ldiv	Divides a long-type integer and calculates the remainder.
malloc	Allocates a memory area.
mblen	Calculates the length of a multibyte character string.
mbstowcs	Converts a multibyte character string into a wide character string.
mbtowc	Converts a multibyte character into a wide character.
qsort	Sorts elements in an array.
realloc	Changes the size of an allocated memory area.
strtod	Converts a character string into a double-type integer.
strtol	Converts a character string into a long-type integer.
strtoul	Converts a character string into an unsigned long-type integer.
wcstombs	Converts a wide character string into a multibyte character string.
wctomb	Converts a wide character into a multibyte character.

string.h

[Function] Declares the prototypes of string handling functions and memory handling functions.

Type	Function	Contents
Copy	strcpy	Copies a character string.
	strncpy	Copies a character string ('n' characters).
Concatenate	strcat	Concatenates character strings.
	strncat	Concatenates character strings ('n' characters).
Compare	strcmp	Compares character strings .
	strcoll	Compares character strings (using locale information).
	stricmp	Compares character strings. (All alphabets are handled as upper-case letters.)
	strncmp	Compares character strings ('n' characters).
	strnicmp	Compares character strings ('n' characters). (All alphabets are handled as upper-case letters.)
Search	strchr	Searches the specified character beginning with the top of the character string.
	strcspn	Calculates the length (number) of unspecified characters that are not found in the other character string.
	strpbrk	Searches the specified character in a character string from the other character string.
	strrchr	Searches the specified character from the end of a character string.
	strspn	Calculates the length (number) of specified characters that are found in the other character string.
	strstr	Searches the specified character from a character string.
	strtok	Divides some character string from a character string into tokens.
Length	strlen	Calculates the number of characters in a character string.
Convert	strerror	Converts an error number into a character string.
	strxfrm	Converts a character string (using locale information).
Initialize	bzero	Initializes a memory area (by clearing it to zero).
Copy	bcopy	Copies characters from a memory area to another.
	memcpy	Copies characters ('n' bytes) from a memory area to another.
	memset	Set a memory area by filling with characters.
Compare	memcmp	Compares memory areas ('n' bytes).
	memicmp	Compares memory areas (with alphabets handled as upper-case letters).
Search	memchr	Searches a character from a memory area.

time.h

[Function] Declares the functions necessary to indicate the current calendar time and defines the type.

E.2 Standard Function Reference

E.2.1 Overview of Standard Library

NC77 has 119 Standard Library items. Each function can be classified into one of the following 11 categories according to its function.

1.String Handling Functions

Functions to copy and compare character strings, etc.

2.Character Handling Functions

Functions to judge letters and decimal characters, etc., and to covert uppercase to lowercase and vice-versa.

3.I/O Functions

Functions to input and output characters and character strings. These include functions for formatted I/O and character string manipulation.

4.Memory Management Functions

Functions for dynamically securing and releasing memory areas.

5.Memory Manipulation Functions

Functions to copy, set, and compare memory areas.

6.Execution Control Functions

Functions to execute and terminate programs, and for jumping from the currently executing function to another function.

7.Mathematical Functions

Functions for calculating sines (sin) and cosines (cos), etc.

* These functions require time.
  Therefore, pay attention to the use of the watchdog timer.

8.Integer Arithmetic Functions

Functions for performing calculations on integer values.

9.Character String Value Convert Functions

Functions for converting character strings to numerical values.

10. Multi-byte Character and Multi-byte Character String Manipulate Functions

Functions for processing multi-byte characters and multi-byte character strings.

11. Locale Functions

Locale-related functions.

E.2.2 List of Standard Library Functions by Function

a. String Handling Functions

The following lists String Handling Functions.

Table E.2 String Handling Functions

Type	Function	Contents	Reentrant
Copy	strcpy	Copies a character string.	❍
	strncpy	Copies a character string ('n' characters).	❍
Concatenate	strcat	Concatenates character strings.	❍
	strncat	Concatenates character strings ('n' characters).	❍
Compare	strcmp	Compares character strings .	❍
	strcoll	Compares character strings (using locale information).	❍
	stricmp	Compares character strings. (All alphabets are handled as upper-case letters.)	❍
	strncmp	Compares character strings ('n' characters).	❍
	strnicmp	Compares character strings ('n' characters). (All alphabets are handled as upper-case letters.)	❍
Search	strchr	Searches the specified character beginning with the top of the character string.	❍
	strcspn	Calculates the length (number) of unspecified characters that are not found in the other character string.	❍
	strpbrk	Searches the specified character in a character string from the other character string.	❍
	strrchr	Searches the specified character from the end of a character string.	❍
	strspn	Calculates the length (number) of specified characters that are found in the other character string.	❍
	strstr	Searches the specified character from a character string.	❍
	strtok	Divides some character string from a character string into tokens.	✕
Length	strlen	Calculates the number of characters in a character string.	❍
Convert	strerror	Converts an error number into a character string.	✕
	strxfrm	Converts a character string (using locale information).	❍

* Several standard functions use global variables that are specific to that function. If, while that function is called and is being executed, an interrupt occurs and that same function is called by the interrupt processing program, the global variables used by the function when first called may be overwritten.

This does not occur to global variables of functions with reentrancy (indicated by a ❍ in the table). However, if the function does not have reentrancy (indicated by a ✕ in the table), care must be taken if the function is also used by an interrupt processing program.

b. Character Handling Functions

The following lists character handling functions.

Table E.3 Character Handling Functions

Function	Contents	Reentrant
isalnum	Checks whether the character is an alphabet or numeral.	❍
isalpha	Checks whether the character is an alphabet.	❍
iscntrl	Checks whether the character is a control character.	❍
isdigit	Checks whether the character is a numeral.	❍
isgraph	Checks whether the character is printable (except a blank).	❍
islower	Checks whether the character is a lower-case letter.	❍
isprint	Checks whether the character is printable (including a blank).	❍
ispunct	Checks whether the character is a punctuation character.	❍
isspace	Checks whether the character is a blank, tab, or new line.	❍
isupper	Checks whether the character is an upper-case letter.	❍
isxdigit	Checks whether the character is a hexadecimal character.	❍
tolower	Converts the character from an upper-case to a lower-case.	❍
toupper	Converts the character from a lower-case to an upper-case.	❍

c. Input/Output Functions

The following lists Input/Output functions.

Table E.4 Input/Output Functions

Type	Function	Contents	Reentrant
Initialize	init	Initializes 7700 series's input/outputs.	❍
	clearerror	Initializes (clears) error status specifiers.	×
Input	fgetc	Inputs one character from the stream.	×
	getc	Inputs one character from the stream.	×
	getchar	Inputs one character from stdin.	×
	fgets	Inputs one line from the stream.	×
	gets	Inputs one line from stdin.	×
	fread	Inputs the specified items of data from the stream.	×
	scanf	Inputs characters with format from stdin.	×
	fscanf	Inputs characters with format from the stream.	×
	sscanf	Inputs data with format from a character string.	×
Output	fputc	Outputs one character to the stream.	×
	putc	Outputs one character to the stream.	×
	putchar	Outputs one character to stdout.	×
	fputs	Outputs one line to the stream.	×
	puts	Outputs one line to stdout.	×
	fwrite	Outputs the specified items of data to the stream.	×
	perror	Outputs an error message to stdout.	×
	printf	Outputs characters with format to stdout.	×
	fflush	Flushes the stream of an output buffer.	×
	fprintf	Outputs characters with format to the stream.	×
	sprintf	Writes text with format to a character string.	×
	vfprintf	Output to a stream with format.	×
	vprintf	Output to stdout with format.	×
	vsprintf	Output to a buffer with format.	×
Return	ungetc	Sends one character back to the input stream.	×
Determi-	ferror	Checks input/output errors.	×
nation	feof	Checks EOF (End of File).	×

d. Memory Management Functions

The following lists memory management functions.

Table E.5 Memory Management Functions

Function	Contents	Reentrant
calloc	Allocates a memory area and initializes it to zero (0).	×
free	Frees the allocated memory area.	×
malloc	Allocates a memory area.	×
realloc	Changes the size of an allocated memory area.	×

e. Memory Handling Functions

The following lists memory handling functions.

Table E.6   Memory Handling Functions

Type	Function	Contents	Reentrant
Initialize	bzero	Initializes a memory area (by clearing it to zero).	❍
Copy	bcopy	Copies characters from a memory area to another.	❍
	memcpy	Copies characters ('n' bytes) from a memory area to another.	❍
	memset	Set a memory area by filling with characters.	❍
Compare	memcmp	Compares memory areas ('n' bytes).	❍
	memicmp	Compares memory areas (with alphabets handled as upper-case letters).	❍
Move	memmove	Moves the area of a character string.	❍
Search	memchr	Searches a character from a memory area.	❍

f. Execution Control Functions

The following lists execution control functions.

Table E.7   Execution Control Functions

Function	Contents	Reentrant
abort	Terminates the execution of the program.	❍
longjmp	Performs a global jump.	❍
setjmp	Sets a stack environment for a global jump.	❍

g. Mathematical Functions

The following lists mathematical functions.

Table E.8 Mathematical Functions

Function	Contents	Reentrant
acos	Calculates arc cosine.	❍
asin	Calculates arc sine.	❍
atan	Calculates arc tangent.	❍
atan2	Calculates arc tangent.	❍
ceil	Calculates an integer carry value.	❍
cos	Calculates cosine.	❍
cosh	Calculates hyperbolic cosine.	❍
exp	Calculates exponential function.	❍
fabs	Calculates the absolute value of a double-precision floating-point number.	❍
floor	Calculates an integer borrow value.	❍
fmod	Calculates the remainder.	❍
frexp	Divides floating-point number into mantissa and exponent parts.	❍
labs	Calculates the absolute value of a long-type integer.	❍
ldexp	Calculates the power of a floating-point number.	❍
log	Calculates natural logarithm.	❍
log10	Calculates common logarithm.	❍
modf	Calculates the division of a real number into the mantissa and exponent parts.	❍
pow	Calculates the power of a number.	❍
sin	Calculates sine.	❍
sinh	Calculates hyperbolic sine.	❍
sqrt	Calculates the square root of a numeric value.	❍
tan	Calculates tangent.	❍
tanh	Calculates hyperbolic tangent.	❍

h. Integer Arithmetic Functions

The following lists integer arithmetic functions.

Table E.9 Integer Arithmetic Functions

Function	Contents	Reentrant
abs	Calculates the absolute value of an integer.	❍
bsearch	Performs binary search in an array.	❍
div	Divides an int-type integer and calculates the remainder.	❍
labs	Calculates the absolute value of a long-type integer.	❍
ldiv	Divides a long-type integer and calculates the remainder.	❍
qsort	Sorts elements in an array.	❍
rand	Generates a pseudo-random number.	❍
srand	Imparts seed to a pseudo-random number generating routine.	❍

i. Character String Value Convert Functions

The following lists character string value convert functions.

Table E.10 Character String Value Convert Functions

Function	Contents	Reentrant
atof	Converts a character string into a double-type floating-point number.	❍
atoi	Converts a character string into an int-type integer.	❍
atol	Converts a character string into a long-type integer.	❍
strtod	Converts a character string into a double-type integer.	❍
strtol	Converts a character string into a long-type integer.	❍
strtoul	Converts a character string into an unsigned long-type integer.	❍

j. Multi-byte Character and Multi-byte Character String Manipulate Functions

The following lists Multibyte Character and Multibyte Character string Manipulate Functions.

Table E.11 Multibyte Character and Multibyte Character String Manipulate Functions

Function	Contents	Reentrant
mblen	Calculates the length of a multibyte character string.	❍
mbstowcs	Converts a multibyte character string into a wide character string.	❍
mbtowc	Converts a multibyte character into a wide character.	❍
wcstombs	Converts a wide character string into a multibyte character string.	❍
wctomb	Converts a wide character into a multibyte character.	❍

k. Localization Functions

The following lists localization functions.

Table E.12 Localization Functions

Function	Contents	Reentrant
localeconv	Initializes struct lconv.	❍
setlocale	Sets and searches the locale information of a program.	❍

E.2.3 Standard Function Reference

The following describes the detailed specifications of the standard functions provided in NC77. The functions are listed in alphabetical order.

Note that the standard header file (extension .h) shown under "Format" must be included when that function is used.

abort

Execution Control Functions

[Function] Terminates the execution of the program abnormally.

[Format] #include <stdlib.h>

void abort(void);

[Method] function

[Variable] No argument used.

[ReturnValue] ● No value is returned.

[Description] ● Terminates the execution of the program abnormally.

[Note] ● Actually,the program loops in the abort function.

abs

Integer Arithmetic Functions

[Function] Calculates the absolute value of an integer.

[Format] #include <stdlib.h>

int abs(n);

[Method] function

[Variable] int n; .... Integer

[ReturnValue] ● Returns the absolute value of integer n (distance from 0).

acos

Mathematical Functions

[Function] Calculates arc cosine.

[Format] #include <math.h>

double _far acos(x);

[Method] function

[Variable] double x; .... arbitrary real number

[ReturnValue]
- ● Assumes an error and returns 0 if the value of given real number x is outside the range of -1.0 to 1.0.
- ● Otherwise, returns a value in the range from 0 to π radian.

asin

Mathematical Functions

[Function] Calculates arc sine.

[Format] #include <math.h>

double _far asin(x);

[Method] function

[Variable] double x; .... arbitrary real number

[ReturnValue]
- ● Assumes an error and returns 0 if the value of given real number x is outside the range of -1.0 to 1.0.
- ● Otherwise, returns a value in the range from -π/2 to π/2 radian.

atan

Mathematical Functions

[Function] Calculates arc tangent.

[Format] #include <math.h>

double _far atan(x);

[Method] function

[Variable] double x; .... arbitrary real number

[ReturnValue] ● Returns a value in the range from -π/2 to π/2 radian.

atan2

Mathematical Functions

[Function] Calculates arc tangent.

[Format] #include <math.h>

double _far atan2(x , y);

[Method] function

[Variable] double x; .... arbitrary real number
double y; .... arbitrary real number

[ReturnValue] ● Returns a value in the range from -π to π radian.

atof

Character String Value Convert Functions

[Function] Converts a character string into a double-type floating- point number.

[Format] #include <stdlib.h>

double _far atof(s);

[Method] function

[Variable] const char * s; .... Pointer to the converted character string

[ReturnValue] ● Returns the value derived by converting a character string into a double-precision floating-point number.

atoi

Character String Convert Functions

[Function] Converts a character string into an int-type integer.

[Format] #include <stdlib.h>

int _far atoi(s);

[Method] function

[Variable] const char * s; .... Pointer to the converted character string

[ReturnValue] ● Returns the value derived by converting a character string into an int-type integer.

atol

Character String Convert Functions

[Function] Converts a character string into a long-type integer.

[Format] #include <stdlib.h>

long _far atol(s);

[Method] function

[Variable] const char * s; .... Pointer to the converted character string

[ReturnValue] ● Returns the value derived by converting a character string into an long-type integer.

bcopy

Memory Handling Functions

[Function] Copies characters from a memory area to another.

[Format] #include <string.h>

void _far bcopy(src, dtop, size);

[Method] function

[Variable] char _far * src; .... Start address of the memory area to be copied from
char _far * dtop; .... Start address of the memory area to be copied to
unsigned long size; Number of bytes to be copied

[ReturnValue] ● No value is returned.

[Description] ● Copies the number of bytes specified in size from the beginning of the area specified in src to the area specified in dtop.
● Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

bsearch

Integer Arithmetic Functions

[Function] Performs binary search in an array.

[Format] #include <stdlib.h>

void _far bsearch(key, base, nelem, size, cmp);

[Method] function

[Variable]
const void * s; .... Search key
const void* s; .... Start address of array
size_t nelem; .... Element number
size_t size; .... Element size
int cmp(); .... Compare function

[ReturnValue]
- Returns a pointer to an array element that equals the search key.
- Returns a NULL pointer if no elements matched.

[Description]
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

[Note]
- The specified item is searched from the array after it has been sorted in ascending order.

bzero

Memory Handling Functions

[Function] Initializes a memory area (by clearing it to zero).

[Format] #include <string.h>

void _far bzero(top, size);

[Method] function

[Argument]
char _far * top; .... Start address of the memory area to be cleared to zero
unsigned long size; .... Number of bytes to be cleared to zero

[ReturnValue]
- No value is returned.

[Description]
- Initializes (to 0) the number of bytes specified in size from the starting address of the area specified in top.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

calloc

Memory Management Functions

[Function] Allocates a memory area and initializes it to zero (0).

[Format] #include <stdlib.h>

void _far * _far calloc(n, size);

[Method] function

[Argument] size_t n; .... Number of elements
size_t size; .... Value indicating the element size in bytes

[ReturnValue]
- Returns start address of allocated memory area.
- Returns NULL if a memory area of the specified size could not be allocated.

[Description]
- After allocating the specified memory, it is cleared to zero.
- The size of the memory area is the product of the two parameters.

[Rule]
- The rules for securing memory are the same as for malloc.

ceil

Mathematical Functions

[Function] Calculates an integer carry value.

[Format] #include <math.h>

double _far ceil(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue]
- Returns the minimum integer value from among integers larger than given real number x.

clearerr

Input/Output Functions

[Function] Initializes (clears) error status specifiers.

[Format] #include <stdio.h>

void _far clearerr(stream);

[Method] function

[Argument] FILE * stream; .... Pointer of stream

[ReturnValue] ● No value is returned.

[Description] ● Resets the error designator and end of file designator to their normal values.

cos

Mathematical Functions

[Function] Calculates cosine.

[Format] #include <math.h>

double _far cos(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the cosine of given real number x handled in units of radian.

cosh

Mathematical Functions

[Function] Calculates hyperbolic cosine.

[Format] #include <math.h>

double _far cosh(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the hyperbolic cosine of given real number x.

div

Integer Arithmetic Functions

[Function] Divides an int-type integer and calculates the remainder.

[Format] #include <stdlib.h>

div_t _far div(number, denom);

[Method] function

[Argument] int number;......................... Dividend
int denom; .... Divisor

[ReturnValue] ● Returns the quotient derived by dividing "number" by "denom" and the remainder of the division.

[Description] ● Returns the quotient derived by dividing "number" by "denom" and the remainder of the division in structure div_t.
● div_t is defined in stdlib.h. This structure consists of members int quot and int rem.

exp

Mathematical Functions

[Function] Calculates exponential function.

[Format] #include <math.h>

double _far exp(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the calculation result of an exponential function of given real number x.

fabs

Mathematical Functions

[Function] Calculates the absolute value of a double-precision floating-point number.

[Format] #include <math.h>

double _far fabs(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the absolute value of a double-precision floating-point number.

feof

Input/Output Functions

[Function] Checks EOF (End of File).

[Format] #include <stdio.h>

int _far feof(stream);

[Method] macro

[Argument] FILE * stream; .... Pointer of stream

[ReturnValue]
- Returns "true" (other than 0) if the stream is EOF.
- Otherwise, returns NULL (0).

[Description]
- Determines if the stream has been read to the EOF.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

ferror

Input/Output Functions

[Function] Checks input/output errors.

[Format] #include <stdio.h>

int _far ferror(stream);

[Method] macro

[Argument] FILE * stream; .... Pointer of stream

[ReturnValue]
- Returns "true" (other than 0) if the stream is in error.
- Otherwise, returns NULL (0).

[Description]
- Determines errors in the stream.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

fflush

Input/Output Functions

[Function] Flushes the stream of an output buffer.

[Format] #include <stdio.h>

int _far fflush(stream);

[Method] function

[Argument] FILE * stream; .... Pointer of stream

[ReturnValue] ● Always returns 0.

fgetc

Input/Output Functions

[Function] Reads one character from the stream.

[Format] #include <stdio.h>

int _far fgetc(stream);

[Method] function

[Argument] FILE * stream; .... Pointer of stream

[ReturnValue]
- ● Returns the one input character.
- ● Returns EOF if an error or the end of the stream is encountered.

[Description]
- ● Reads one character from the stream.
- ● Interprets code 0x1A as the end code and ignores any subsequent data.
- ● To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

fgets

Input/Output Functions

[Function] Reads one line from the stream.

[Format]

```
#include     <stdio.h>

char * _far fgets( buffer, n, stream );
```

[Method] function

[Argument]

char * buffer; .... Pointer of the location to be stored in
int n; .... Maximum number of characters
FILE * stream; .... Pointer of stream

[ReturnValue]

- Returns the pointer of the location to be stored (the same pointer as given by the argument) if normally input.
- Returns the NULL pointer if an error or the end of the stream is encountered.

[Description]

- Reads character string from the specified stream and stores it in the buffer
- Input ends at the input of any of the following:
  - new line character ('\n')
  - n-1 characters
  - end of stream
- A null character ('\0') is appended to the end of the input character string.
- The new line character ('\n') is stored as-is.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- To process the parameter using the far pointer for buffer or stream, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

floor

Mathematical Functions

[Function] Calculates an integer borrow value.

[Format] #include <math.h>

double _far floor(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● The real value is truncated to form an integer,which is returned as a double type.

fmod

Mathematical Functions

[Function] Calculates the remainder.

[Format] #include <math.h>

double _far fmod(x ,y);

[Method] function

[Argument] double x; .... dividend
double y; .... divisor

[ReturnValue] ● Returns a remainder that derives when dividend x is divided by divisor y.

fprintf

Input/Output Functions

[Function] Outputs characters with format to the stream.

[Format] #include <stdio.h>

int _far fprintf(stream, format, argument...);

[Method] function

[Argument] FILE * stream; .... Pointer of stream
const char * format;. .... Pointer of the format specifying character string

[ReturnValue]
- Returns the number of characters output.
- Returns EOF if a hardware error occurs.

[Description]
- Argument is converted to a character string according to format and output to the stream.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- Format is specified in the same way as in printf.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

fputc

Input/Output Functions

[Function] Outputs one character to the stream.

[Format] #include <stdio.h>

int _far fputc(c, stream);

[Method] function

[Argument] int c; .... Character to be output
FILE * stream; .... Pointer of the stream

[ReturnValue]
- Returns the output character if output normally.
- Returns EOF if an error occurs.

[Description]
- Outputs one character to the stream.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

fputs

Input/Output Functions

[Function] Outputs one line to the stream.

[Format] #include <stdio.h>

int _far fputs (str, stream);

[Method] function

[Argument] const char * str; .... Pointer of the character string to be output
FILE * stream; .... Pointer of the stream

[ReturnValue]
- ● Returns 0 if output normally.
- ● Returns any value other than 0 (EOF) if an error occurs.

[Description]
- ● Outputs one line to the stream.
- ● Interprets code 0x1A as the end code and ignores any subsequent data.
- ● To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

fread

Input/Output Functions

[Function] Reads fixed-length data from the stream

[Format] #include <stdio.h>

size_t _far fread(buffer, size, count, stream);

[Method] function

[Argument] void * buffer; .... Pointer of the location to be stored in
size_t size; .... Number of bytes in one data item
size_t count; .... Maximum number of data items
FILE * stream; .... Pointer of stream

[ReturnValue]
- ● Returns the number of data items input.

[Description]
- ● Reads data of the size specified in size from the stream and stores it in the buffer. This is repeated by the number of times specified in count.
- ● If the end of the stream is encountered before the data specified in count has been input, this function returns the number of data items read up to the end of the stream.
- ● Interprets code 0x1A as the end code and ignores any subsequent data.
- ● To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

free

Memory Management Function

[Function] Frees the allocated memory area.

[Format] #include   <stdlib.h>

void _far free(cp);

[Method] function

[Argument] void _far * cp; .... Pointer to the memory area to be freed

[ReturnValue] ● No value is returned.

[Description] ● Frees memory areas previously allocated with malloc or calloc.
● No processing is performed if you specify NULL in the parameter.

frexp

Mathematical Functions

[Function] Divides floating-point number into mantissa and exponent parts.

[Format] #include   <math.h>

double _far frexp(x,prexp);

[Method] function

[Argument] double x; .... float-point number
int * prexp; .... Pointer to an area for storing a 2-based exponent

[ReturnValue] ● Returns the floating-point number x mantissa part.

fscanf

Input/Output Function

[Function] Reads characters with format from the stream.

[Format] #include <stdio.h>

int _far fscanf(stream, format, argument...);

[Method] function

[Argument] FILE * stream; .... Pointer of stream
const char * format; .... Pointer of the input character string

[ReturnValue]
- Returns the number of data entries stored in each argument.
- Returns EOF if EOF is input from the stream as data.

[Description]
- Converts the characters input from the stream as specified in format and stores them in the variables shown in the arguments.
- Argument must be a pointer to the respective variable.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- Format is specified in the same way as in scanf.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

fwrite

Input/Output Functions

[Function] Outputs the specified items of data to the stream.

[Format] #include <stdio.h>

size_t _far fwrite(buffer, size, count, stream);

[Method] function

[Argument] const void * buffer; .... Pointer of the output data
size_t size; .... Number of bytes in one data item
size_t count; .... Maximum number of data items
FILE * stream; .... Pointer of the stream

[ReturnValue]
- Returns the number of data items output.

[Description]
- Outputs data with the size specified in size to the stream. Data is output by the number of times specified in count.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- If an error occurs before the amount of data specified in count has been input, this function returns the number of data items output to that point.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

getc

Input/Output Functions

[Function] Reads one character from the stream.

[Format] #include <stdio.h>

int _far getc(stream);

[Method] macro

[Argument] FILE * stream; .... Pointer of stream

[ReturnValue]
- Returns the one input character.
- Returns EOF if an error or the end of the stream is encountered.

[Description]
- Reads one character from the stream.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

getchar

Input/Output Functions

[Function] Reads one character from stdin.

[Format] #include <stdio.h>

int _far getchar(void);

[Method] macro

[Argument] No argument used.

[ReturnValue]
- Returns the one input character.
- Returns EOF if an error or the end of the file is encountered.

[Description]
- Reads one character from stream(stdin).
- Interprets code 0x1A as the end code and ignores any subsequent data.

gets

Input/Output Functions

[Function] Reads one line from stdin.

[Format] #include <stdio.h>

char * _far gets(buffer);

[Method] function

[Argument] char * buffer; .... Pointer of the location to be stored in

[ReturnValue]
- Returns the pointer of the location to be stored (the same pointer as given by the argument) if normally input.
- Returns the NULL pointer if an error or the end of the file is encountered.

[Description]
- Reads character string from stdin and stores it in the buffer.
- The new line character ('\n') at the end of the line is replaced with the null character ('\0').
- Interprets code 0x1A as the end code and ignores any subsequent data.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

init

Input/Output Functions

[Function] Initializes 7700 Family's input/outputs.

[Format] #include <stdio.h>

void _far init(void);

[Method] function

[Argument] No argument used.

[ReturnValue]
- No value is returned.

[Description]
- Initializes the stream. Also calls speed and init_prn in the function to make the initial settings of the UART and Centronics output device.
- init is normally used by calling it from the startup program.

isalnum

Character Handling Functions

[Function] Checks whether the character is an alphabet or numeral (A - Z,a - z,0 - 9).

[Format] #include <ctype.h>

int isalnum(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if an alphabet or numeral.
- Returns 0 if not an alphabet nor numeral.

[Description]
- Determines the type of character in the parameter.

isalpha

Character Handling Functions

[Function] Checks whether the character is an alphabet (A - Z,a - z).

[Format] #include <ctype.h>

int isalpha(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if an alphabet.
- Returns 0 if not an alphabet.

[Description]
- Determines the type of character in the parameter.

isclntrl

Character Handling Functions

[Function] Checks whether the character is a control character (0x00 - 0x1f,0x7f).

[Format] #include <ctype.h>

int iscntrl(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if a numeral.
- Returns 0 if not a control character.

[Description]
- Determines the type of character in the parameter.

isdigit

Character Handling Functions

[Function] Checks whether the character is a numeral(0 - 9).

[Format] #include <ctype.h>

int isdigit(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if a numeral.
- Returns 0 if not a numeral.

[Description]
- Determines the type of character in the parameter.

isgraph

Character Handling Functions

[Function] Checks whether the character is printable (except a blank)(0x21 - 0x7e).

[Format] #include <ctype.h>

int isgraph(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if printable.
- Returns 0 if not printable.

[Description]
- Determines the type of character in the parameter.

islower

Character Handling Functions

[Function] Checks whether the character is a lower-case letter(a - z).

[Format] #include <ctype.h>

int islower(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if a lower-case letter.
- Returns 0 if not a lower-case letter.

[Description] Determines the type of character in the parameter.

isprint

Character Handling Functions

[Function] Checks whether the character is printable (including a blank)(0x20 - 0x7e).

[Format] #include <ctype.h>

int isprint(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if printable.
- Returns 0 if not printable.

[Description]
- Determines the type of character in the parameter.

ispunct

Character Handling Functions

[Function] Checks whether the character is a punctuation character.

[Format] #include <ctype.h>

int ispunct(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if a punctuation character.
- Returns 0 if not a punctuation character.

[Description]
- Determines the type of character in the parameter.

isspace

Character Handling Functions

[Function] Checks whether the character is a blank, tab, or new line.

[Format] #include <ctype.h>

int isspace(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if a blank, tab, or new line.
- Returns 0 if not a blank, tab, or new line.

[Description]
- Determines the type of character in the parameter.

isupper

Character Handling Functions

[Function] Checks whether the character is an upper-case letter(A - Z).

[Format] #include <ctype.h>

int isupper(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if an upper character.
- Returns 0 if not an upper-case letter.

[Description]
- Determines the type of character in the parameter.

isxdigit

Character Handling Functions

[Function] Checks whether the character is a hexadecimal character(0 - 9,A - F,a - f).

[Format] #include <ctype.h>

int isxdigit(c);

[Method] macro

[Argument] int c; .... Character to be checked

[ReturnValue]
- Returns any value other than 0 if a hexadecimal character.
- Returns 0 if not a hexadecimal character.

[Description]
- Determines the type of character in the parameter.

labs

Integer Arithmetic Functions

[Function] Calculates the absolute value of a long-type integer.

[Format] #include <stdlib.h>

long _far labs(n);

[Method] function

[Argument] long n; .... Long integer

[ReturnValue]
- Returns the absolute value of a long-type integer (distance from 0).

ldexp

Localization Functions

[Function] Calculates the power of a floating-point number.

[Format] #include <math.h>

double _far ldexp(x,exp);

[Method] function

[Argument] double x; .... Float-point number
int exp; .... Power of number

[ReturnValue] ● Returns x* (exp power of 2).

ldiv

Integer Arithmetic Functions

[Function] Divides a long-type integer and calculates the remainder.

[Format] #include <stdlib.h>

ldiv_t _far ldiv(number, denom);

[Method] function

[Argument] long number; .... Dividend
long denom; .... Divisor

[ReturnValue] ● Returns the quotient derived by dividing "number" by "denom" and the remainder of the division.

[Description] ● Returns the quotient derived by dividing "number" by "denom" and the remainder of the division in the structure ldiv_t.
● ldiv_t is defined in stdlib.h. This structure consists of members long quot and long rem.

localeconv

Localization Functions

[Function] Initializes struct lconv.

[Format] #include <locale.h>

struct lconv _far * _far localeconv(void);

[Method] function

[Argument] No argument used.

[ReturnValue] ● Returns a pointer to the initialized struct lconv.

log

Mathematical Functions

[Function] Calculates natural logarithm.

[Format] #include <math.h>

double _far log(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the natural logarithm of given real number x.

[Description] ● This is the reverse function of exp.

log10

Mathematical Functions

[Function] Calculates common logarithm.

[Format] #include <math.h>

double _far log10(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the common logarithm of given real number x.

longjmp

Execution Control Functions

[Function] Restores the environment when making a function call

[Format] #include <setjmp.h>

void _far longjmp(env, val);

[Method] function

[Argument] jmp_buf _far * env; .... Pointer to the area where environment is restored
int val; .... Value returned as a result of setjmp

[ReturnValue] ● No value is returned.

[Description]
- ● Restores the environment from the area indicated in "env".
- ● Program control is passed to the statement following that from which setjmp was called.
- ● The value specified in "value" is returned as the result of setjmp. However, if "val" is "0", it is converted to "1".

malloc

Memory Management Functions

[Function] Allocates a memory area.

[Format] #include <stdlib.h>

void _far * _far malloc(nbytes);

[Method] function

[Argument] size_t nbytes; .... Size of memory area (in bytes) to be allocated

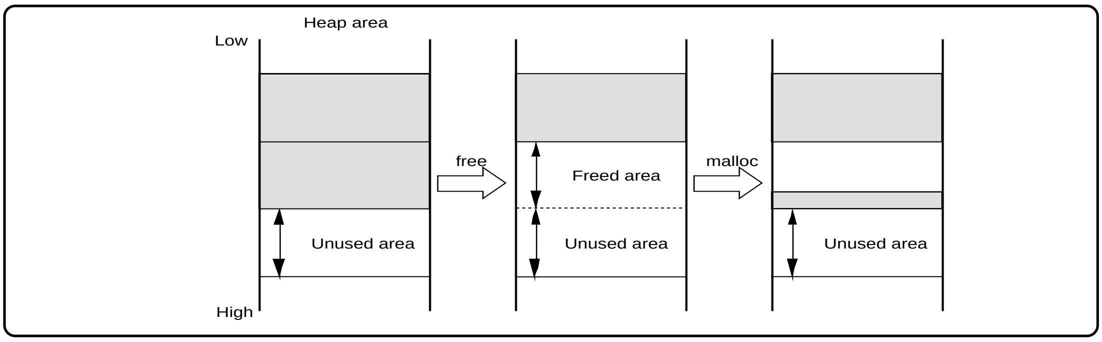
[ReturnValue] ● Returns NULL if a memory area of the specified size could not be allocated.

[Description] ● Dynamically allocates memory areas

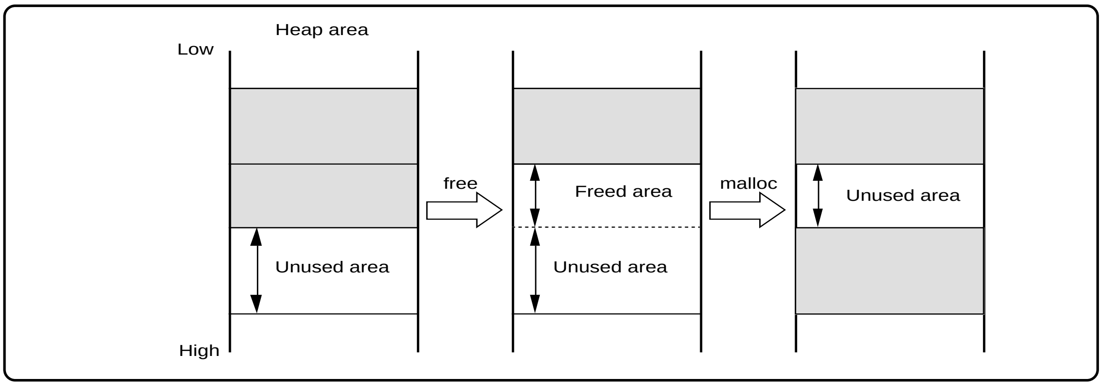
[Rule] ● malloc performs the following two checks to secure memory in the appropriate location.

(1)If memory areas have been freed with free

(1-1)If the amount of memory to be secured is smaller than that freed, the area is secured from the high address of the contiguously empty area created by free toward the low address.



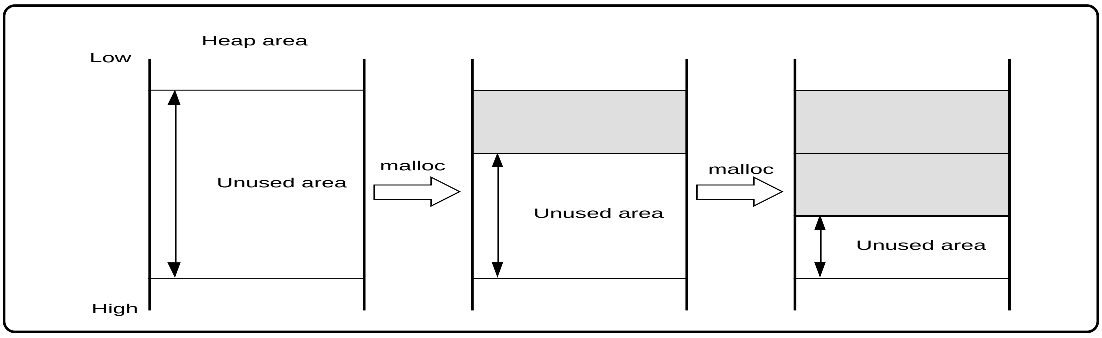
(1-2)If the amount of memory to be secured is larger than that freed, the area is secured from the lowest address of the unused memory toward the high address.



malloc

(2)If no memory area has been freed with free

(2-1)If there is any unused area that can be secured, the area is secured from the lowest address of the unused memory toward the high address.



(2-2)If there is no unused area that can be secured, malloc returns NULL without any memory being secured.

[Note] No garbage collection is performed. Therefore, even if there are lots of small unused portions of memory, no memory is secured and malloc returns NULL unless there is an unused portion of memory that is larger than the specified size.

mblen

Multi-byte Character Multi-byte Character String Manipulate Functions

[Function] Calculates the length of a multi-byte character string.

[Format] #include <stdlib.h>

int _far mblen (s,n);

[Method] function

[Argument] const char * s; .... Pointer to a multi-byte character string
size_t n; .... Number of searched byte

[ReturnValue]
- ● Returns the number of bytes in the character string if 's' configures a correct multi-byte character string.
- ● Returns -1 if 's' does not configure a correct multi-byte character string.
- ● Returns 0 if 's' indicates a NULL character.

mbstowcs

Multi-byte Character Multi-byte Character String Manipulate Functions

[Function] Converts a multi-byte character string into a wide character string.

[Format] #include <stdlib.h>

size_t _far mbstowcs(wcs,s,n);

[Method] function

[Argument]
wchar_t * wcs; .... Pointer to an area for storing conversion wide character string
const char * s; .... Pointer to a multi-byte character string
size_t n; .... Number of wide characters stored

[ReturnValue]
- Returns the number of characters in the converted multi-byte character string.
- Returns -1 if 's' does not configure a correct multi-byte character string.

mbtowc

Multi-byte Character Multi-byte Character String Manipulate Functions

[Function] Converts a multi-byte character into a wide character.

[Format] #include <stdlib.h>

int _far mbtowc(wcs,s,n);

[Method] function

[Argument]
wchar_t * wcs; .... Pointer to an area for storing conversion wide character string
const char * s; .... Pointer to a multi-byte character string

size_t n; .... Number of wide characters stored

[ReturnValue]
- Returns the number of wide characters converted if 's' configure a correct multi-byte character string.
- Returns -1 if 's' does not configure a correct multi-byte character string.
- Returns 0 if 's' indicates a NULL character.

memchr

Memory Handling Functions

[Function] Searches a character from a memory area.

[Format] #include <string.h>

void _far * _far memchr(s, c, n);

[Method] function

[Argument] const void _far * s; .... Pointer to the memory area to be searched from
int c; .... Character to be searched
size_t n; .... Size of the memory area to be searched

[ReturnValue]
- Returns the position (pointer) of the specified character "c" where it is found.
- Returns NULL if the character "c" could not be found in the memory area.

[Description]
- Searches for the characters shown in "c" in the amount of memory specified in "n" starting at the address specified in "s".
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

memcmp

Memory Handling Functions

[Function] Compares memory areas ('n' bytes).

[Format] #include <string.h>

int _far memcmp(s1, s2, n);

[Method] function

[Argument] const void _far * s1; .... Pointer to the first memory area to be compared
const void _far * s2; .... Pointer to the second memory area to be compared
size_t n; .... Number of bytes to be compared

[ReturnValue]
- Return Value==0 .... The two memory areas are equal.
- Return Value>0 .... The first memory area (s1) is greater than the other.
- Return Value<0 .... The second memory area (s2) is greater than the other.

[Description]
- Compares each of n bytes of two memory areas
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

memcpy

Memory Handling Functions

[Function] Copies n bytes of memory

[Format] #include <string.h>

void _far * _far memcpy(s1, s2, n);

[Method] function

[Argument]
void _far * s1; .... Pointer to the memory area to be copied to
const void _far * 2; .... Pointer to the memory area to be copied from
size_t n; .... Number of bytes to be copied

[ReturnValue]
- Returns the pointer to the memory area to which the characters have been copied.

[Description]
- Copies "n" bytes from memory "S2" to memory "S1".
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

memicmp

Memory Handling Functions

[Function] Compares memory areas (with alphabets handled as upper-case letters).

[Format] #include <string.h>

int _far memicmp(s1, s2, n);

[Method] function

[Argument]
char _far * s1; .... Pointer to the first memory area to be compared
char _far * s2; .... Pointer to the second memory area to be compared
size_t n; .... Number of bytes to be compared

[ReturnValue]
- Return Value==0 .... The two memory areas are equal.
- Return Value>0 .... The first memory area (s1) is greater than the other.
- Return Value<0 .... The second memory area (s2) is greater than the other.

[Description]
- Compares memory areas (with alphabets handled as upper-case letters).
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

memmove

Memory Handling Functions

[Function] Moves the area of a character string.

[Format] #include <string.h>

void _far * _far memmove(s1, s2, n);

[Method] function

[Argument] void * s1; .... Pointer to be moved to
const void * s2; .... Pointer to be moved from
size_t n; .... Number of bytes to be moved

[ReturnValue] ● Returns a pointer to the destination of movement.

[Description] ● When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

memset

Memory Handling Functions

[Function] Set a memory area.

[Format] #include <string.h>

char _far* _far memset(s, c, n);

[Method] function

[Argument] void _far * s; .... Pointer to the memory area to be set at
int c; .... Data to be set
size_t n; .... Number of bytes to be set

[ReturnValue] ● Returns the pointer to the memory area which has been set.

[Description] ● Sets "n" bytes of data "c" in memory "s".
● Specifying the -DNEAR_LIB* 1 option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
● When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

modf

Mathematical Functions

[Function] Calculates the division of a real number into the mantissa and exponent parts.

[Format] #include <math.h>

double _far modf (val,pd);

[Method] function

[Argument] double val; .... arbitrary real number
double * pd; .... Pointer to an area for storing an integer

[ReturnValue] ● Returns the decimal part of a real number.

perror

Input/Output Functions

[Function] Outputs an error message to stderr.

[Format] #include <stdio.h>

void _far perror(s);

[Method] function

[Argument] const char * s; .... Pointer to a character string attached before a mes sage.

[ReturnValue] ● No value is returned.

pow

Mathematical Functions

[Function] Calculates the power of a number.

[Format] #include <math.h>

double _far pow(x,y);

[Method] function

[Argument] double x; .... multiplicand
double y; .... multiplier

[ReturnValue] ● Returns the multiplicand x raised to the power of y.

printf

Input/Output Functions

[Function] Outputs characters with format to stdout.

[Format] #include <stdio.h>

int _far printf(format, argument...);

[Method] function

[Argument] const char * format; .... Pointer of the format specifying character string

The part after the percent (%) sign in the character string given in format has the following meaning. The part between [and] is optional. Details of the format are shown below.

Format: %[flag][minimum field width][precision][modifier (l, L, or h)] conversion specification character

Example format: %-05.8ld

[ReturnValue] ● Returns the number of characters output.
● Returns EOF if a hardware error occurs.

[Description] ● Converts argument to a character string as specified in format and outputs the character string to stdout.
● Interprets code 0x1A as the end code and ignores any subsequent data.
● To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

Specifying format in printf-format

1.Conversion specification symbol

- d, i
  Converts the integer in the parameter to a signed decimal.
- u
  Converts the integer in the parameter to an unsigned decimal.
- o
  Converts the integer in the parameter to an unsigned octal.
- x
  Converts the integer in the parameter to an unsigned hexadecimal. Lowercase "abcdef" are equivalent to 0AH to 0FH.
- X
  Converts the integer in the parameter to an unsigned hexadecimal. Uppercase "ABCDEF" are equivalent to 0AH to 0FH.
- c
  Outputs the parameter as an ASCII character.
- s
  Converts the parameter after the string pointer (char *) (and up to a null character '/0' or the precision) to a character string. Note that wchar_t type character strings cannot be processed.
- p
  Outputs the parameter pointer (all types) in the format data bank register and offset. (Example: 00:1205)
- n
  Stores the number of characters output in the integer pointer of the parameter. The parameter is not converted.
- e
  Converts a double-type parameter to the exponent format. The format is [-]d.dddddde±dd.
- E
  Same as e, except that E is used in place of e for the exponent.
- f
  Converts double parameters to [-]d.dddddd format.
- g
  Converts double parameters to the format specified in e or f. Normally, f conversion, but conversion to e type when the exponent is -4 or less or the precision is less than the value of the exponent.
- G
  Same as g except that E is used in place of e for the exponent.

Specifying format in printf-form

2.Flags

● -

Left-aligns the result of conversion in the minimum field width. The default is right alignment.

● +

Adds + or - to the result of signed conversion. By default, only the - is added to negative numbers.

● Blank'   '

By default, a blank is added before the value if the result of signed conversion has no sign.

● #

Adds 0 to the beginning of o conversion.
Adds 0x or 0X to the beginning when other than 0 in x or X conversion.
Always adds the decimal point in e, E, and f conversion.
Always adds the decimal point in g and G conversion and also outputs any 0s in the decimal place.

3.Minimum field width

- Specifies the minimum field width of positive decimal integers.
- When the result of conversion has fewer characters than the specified field width, the left of the field is padded.
- The default padding character is the blank. However, '0' is the padding character if you specified the field with using an integer preceded by '0'.
- If you specified the - flag, the result of conversion is left aligned and padding characters (always blanks) inserted to the right.
- If you specified the asterisk (*) for the minimum field width, the integer in the parameter specifies the field width. If the value of the parameter is negative, the value after the -flag is the positive field width.

4.Precision

Specify a positive integer after '.'. If you specify only '.' with no value, it is interpreted as zero. The function and default value differs according to the conversion type.

Floating point type data is output with a precision of 6 by default. However, no decimal places are output if you specify a precision of 0.

- d, i, o, u, x, and X conversion
- If the number of columns in the result of conversion is less than the specified number, the beginning is padded with zeros.
- If the specified number of columns exceeds the minimum field width, the specified number of columns takes precedence.
- If the number of columns in the specified precision is less than the minimum field width , the field width is processed after the minimum number of columns have been processed.
- The default is 1.
- Nothing is output if zero with converted by zero minimum columns.

Specifying format in printf-form

- s conversion
- Represents the maximum number of characters.
- If the result of conversion exceeds the specified number of characters, the remainder is discarded.
- There is no limit to the number of characters in the default.
- If you specify an asterisk (*) for the precision, the integer of the parameter specifies the precision.
- If the parameter is a negative value, specification of the precision is invalid.
- e, E, and f conversion
- n (where n is the precision) numerals are output after the decimal point.
- g and G conversion
- Valid characters in excess of n (where n is the precision) are not output.

5.l, L or h

- l: d, i, o, u, x, X, and n conversion is performed on long int and unsigned long int parameters.
- h: d, i, o, u, x, and X conversion is performed on short int and unsigned short int parameters.
- If l or h are specified in other than d, i, o, u, x, X, or n conversion, they are ignored.
- L: e, E, f, g, and G conversion is performed on double parameters. *1

* 1.In the standard C specifications,variables e,E,f, and g conversions are performed in the case of L on long double parameters .In NC77/NC79 ,long double types are processed as double types. Therefore, if you specify L, the parameters are processed as double types.

putc

Input/Output Functions

[Function] Outputs one character to the stream.

[Format]

```
#include <stdio.h>

int _far putc( c, stream );
```

[Method] macro

[Argument] int c; .... Character to be output
FILE* stream; .... Pointer of the stream

[ReturnValue]
- Returns the output character if output normally.
- Returns EOF if an error occurs.

[Description]
- Outputs one character to the stream.
- Interprets code 0x1A as the end code and ignores any subsequent data.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

putchar

Input/Output Functions

[Function] Outputs one character to stdout.

[Format]

```
#include <stdio.h>

int _far putchar( c );
```

[Method] macro

[Argument] int c; .... Character to be output

[ReturnValue]
- Returns the output character if output normally.
- Returns EOF if an error occurs.

[Description]
- Outputs one character to stdout.
- Interprets code 0x1A as the end code and ignores any subsequent data

puts

Input/Output Functions

[Function] Outputs one line to stdout.

[Format]
```
#include <stdio.h>

int _far puts( str );
```

[Method] macro

[Argument] char * str;............................ Pointer of the character string to be output

[ReturnValue]
- Returns 0 if output normally.
- Returns -1 (EOF) if an error occurs.

[Description]
- Outputs one line to stdout.
- The null character ('\0') at the end of the character string is replaced with the new line character('n').
- Interprets code 0x1A as the end code and ignores any subsequent data.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

qsort

Integer Arithmetic Functions

[Function] Sorts elements in an array.

[Format]
```
#include <stdlib.h>

void _far qsort( base,nelen,size,cmp( e1,e2) );
```

[Method] function

[Argument]
void * base; .... Start address of array
size_t nelen; .... Element number
size_t size;......................... Element size
int * cmp( ); . ... Compare function

[ReturnValue]
- No value is returned.

[Description]
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

rand

Integer Arithmetic Functions

[Function] Generates a pseudo-random number.

[Format] #include <stdlib.h>

int _far rand(void);

[Method] function

[Argument] No argument used.

[Returnvalue]
- Returns the seed random number series specified in srand.
- The generated random number is a value between 0 and RAND_MAX.

realloc

Memory Management Functions

[Function] Changes the size of an allocated memory area.

[Format] #include <stdlib.h>

void _far * _far realloc(cp, nbytes);

[Method] function

[Argument] void _far * cp; .... Pointer to the memory area before change
size_t nbytes; .... Size of memory area (in bytes) to be changed

[ReturnValue]
- Returns the pointer of the memory area which has had its size changed.
- Returns NULL if a memory area of the specified size could not be secured.

[Description]
- Changes the size of an area already secured using malloc or calloc.
- Specify a previously secured pointer in parameter "cp" and specify the number of bytes to change in "nbytes".

scanf

Input/Output Functions

[Function] Reads characters with format from stdin.

[Format]

```
#include     <stdio.h>
#include     <ctype.h>

int _far scanf( format, argument... );
```

[Method] function

[Argument] char * format; .... Pointer of format specifying character string

The part after the percent (%) sign in the character string given in format has the following meaning. The part between [and] is optional. Details of the format are shown below.

Format: %[*][maximum field width] [modifier (l, L, or h)]conversion specification character

Example format: %*5ld

[ReturnValue]

- Returns the number of data entries stored in each argument.
- Returns EOF if EOF is input from stdin as data.

[Description]

- Converts the characters read from stdin as specified in format and stores them in the variables shown in the arguments.
- Argument must be a pointer to the respective variable.
- The first space character is ignored except in c and [] conversion.
- Interprets code 1A16 as the end code and ignores any subsequent data.
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

Specifying format in scanf-form

1.Conversion specification symbol

- d

  Converts a signed decimal. The target parameter must be a pointer to an integer.
- i

  Converts signed decimal, octal, and hexadecimal input. Octals start with 0. Hexadecimals start with 0x or 0X. The target parameter must be a pointer to an integer.
- u

  Converts an unsigned decimal. The target parameter must be a pointer to an unsigned integer.
- o

  Converts a signed octal. The target parameter must be a pointer to an integer.
- x, X

  Converts a signed hexadecimal. Uppercase or lowercase can be used for 0AH to 0FH. The leading 0x is not included. The target parameter must be a pointer to an integer.
- s

  Stores character strings ending with the null character '\0'. The target parameter must be a pointer to a character array of sufficient size to store the character string including the null character '\0'.

  If input stops when the maximum field width is reached, the character string stored consists of the characters to that point plus the ending null character.
- c

  Stores a character. Space characters are not skipped. If you specify 2 or more for the maximum field width, multiple characters are stored. However, the null character '\0' is not included. The target parameter must be a pointer to a character array of sufficient size to store the character string.
- p

  Converts input in the format data bank register plus offset (Example: 00:1205). The target parameter is a pointer to all types.
- []

  Stores the input characters while the one or more characters between [and] are input. Storing stops when a character other than those between [and] is input. If you specify the circumflex (^) after [, only character other than those between the circumflex and] are legal input characters. Storing stops when one of the specified characters is input.

  The target parameter must be a pointer to a character array of sufficient size to store the character string including the null character '\0', which is automatically added.
- n

  Stores the number of characters already read in format conversion. The target parameter must be a pointer to an integer.
- e, E, f, g, and G

  Convert to floating point format. If you specify modifier l, the target parameter must be a pointer to a double type. The default is a pointer to a float type.

Specifying format in scanf-form

2.* (prevents data storage)

Specifying the asterisk (*) prevents the storage of converted data in the parameter.

3.Maximum field width

- Specify the maximum number of input characters as a positive decimal integer. In any one format conversion, the number of characters read will not exceed this number.
- If, before the specified number of characters has been read, a space character (a character that is true in function isspace()) or a character other than in the specified format is input, reading stops at that character.

4.l, L or h

- l: The results of d, i, o, u, and x conversion are stored as long int and unsigned long int. The results of e, E, f, g, and G conversion are stored as double.
- h: The results of d, i, o, u, and x conversion are stored as short int and unsigned short int.
  - If l or h are specified in other than d, i, o, u, or x conversion, they are ignored.
  - L: The results of e, E, f, g, and G conversion are stored as float.

setjmp

Execution Control Functions

[Function] Saves the environment before a function call

[Format] #include <setjmp.h>

int _far setjmp(env);

[Method] function

[Argument] jmp_buf _far * env; .... Pointer to the area where environment is saved

[ReturnValue] ● Returns the numeric value given by the argument of longjmp.

[Description] ● Saves the environment to the area specified in "env".

setlocale

Localization Functions

[Function] Sets and searches the locale information of a program.

[Format] #include <locale.h>

char _far * _far setlocale(category,locale);

[Method] function

[Argument] int category; .... Locale information, search section information
const char * locale; .... Pointer to a locale information character string

[ReturnValue] ● Returns a pointer to a locale information character string.
● Returns NULL if information cannot be set or searched.

sin

Mathematical Functions

[Function] Calculates sine.

[Format] #include <math.h>

double _far sin(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the sine of given real number x handled in units of radian.

sinh

Mathematical Functions

[Function] Calculates hyperbolic sine.

[Format] #include <math.h>

double _far sinh(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the hyperbolic sine of given real number x.

sprintf

Input/Output Functions

[Function] Writes text with format to a character string.

[Format] int _far sprintf(pointer, format, argument...);

[Method] function

[Argument] char * pointer; .... Pointer of the location to be stored
const char * format; .... Pointer of the format specifying character string

[ReturnValue] ● Returns the number of characters output.

[Description] ● Converts argument to a character string as specified in format and stores them from the pointer.
● Interprets code 0x1A as the end code and ignores any subsequent data.
● Format is specified in the same way as in printf.
● To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

sqrt

Mathematical Functions

[Function] Calculates the square root of a numeric value.

[Format] #include <math.h>

double _far sqrt(x);

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the square root of given real number x.

srand

Integer Arithmetic Functions

[Function] Imparts seed to a pseudo-random number generating routine.

[Format] #include <stdlib.h>

void _far srand(seed);

[Method] function

[Argument] unsigned int seed; .... Series value of random number

[ReturnValue] ● No value is returned.

[Description] ● Initializes (seeds) the pseudo random number series produced by rand using seed.

sscanf

Input/Output Functions

[Function] Reads data with format from a character string.

[Format] #include <stdio.h>

int _far sscanf(string, format, argument...);

[Method] function

[Argument] const char * string; .... Pointer of the input character string
const char * format; .... Pointer of the format specifying character string

[ReturnValue] ● Returns the number of data entries stored in each argument.
● Returns EOF if null character ('/0') is input as data.

[Description] ● Converts the characters input as specified in format and stores them in the variables shown in the arguments.
● Argument must be a pointer to the respective variable.
● Format is specified in the same way as in scanf.
● Interprets code 0x1A as the end code and ignores any subsequent data.
● To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

strcat

String Handling Functions

[Function] Concatenates character strings.

[Format] #include <string.h>

char _far * _far strcat(s1, s2);

[Method] function

[Argument] char _far * s1; .... Pointer to the character string to be concatenated to
char _far * s2; .... Pointer to the character string to be concatenated from

[ReturnValue]
- Returns a pointer to the concatenated character string area(s1).

[Description]
- Concatenates character strings "s1" and "s2" in the sequence s1+s2[*1].
- The concatenated string ends with NULL.
- Specifying the -DNEAR_LIB[*2] option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strchr

String Handling Functions

[Function] Searches the specified character beginning with the top of the character string.

[Format] #include <string.h>

char _far * _far strchr(s, c);

[Method] function

[Argument] const char _far * s; .... Pointer to the character string to be searched in
int c; .... Character to be searched for

[ReturnValue]
- Returns the position of character "c" that is first encountered in character string "s."
- Returns NULL when character string "s" does not contain character "c".

[Description]
- Searches for character "c" starting from the beginning of area "s".
- You can also search for '\0'.
- Specifying the -DNEAR_LIB[*2] option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* 1.There must be adequate space to accommodate s1 plus s2.
* 2.NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strcmp

String Handling Functions

[Function] Compares character strings .

[Format] #include <string.h>

int _far strcmp(s1, s2);

[Method] function

[Argument] const char _far * s1; .... Pointer to the first character string to be compared
const char _far * s2; .... Pointer to the second character string to be compared

[ReturnValue]
- ● ReturnValue==0 .... The two character strings are equal.
- ● ReturnValue>0 .... The first character string (s1) is greater than the other.
- ● ReturnValue<0 .... The second character string (s2) is greater than the other.

[Description]
- ● Compares each byte of two character strings ending with NULL
- ● Specifying the -DNEAR_LIB* option when compiling selects the high-speed library which processes the pointer parameters of this function as having the near attribute.
- ● When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strcoll

String Handling Functions

[Function] Compares character strings (using locale information).

[Format] #include <string.h>

int _far strcoll(s1, s2);

[Method] function

[Argument] const char _far * s1; .... Pointer to the first character string to be compared
const char _far * s2; .... Pointer to the second character string to be compared

[ReturnValue]
- ● ReturnValue==0 .... The two character strings are equal
- ● ReturnValue>0 .... The first character string (s1) is greater than the other
- ● ReturnValue<0 .... The second character string (s2) is greater than the other

[Description]
- ● When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strcpy

String Handling Functions

[Function] Copies a character string.

[Format]
```
#include <string.h>

char _far * _far strcpy( s1, s2 );
```

[Method] function

[Argument]
char _far * s1; .... Pointer to the character string to be copied to
const char _far * s2; .... Pointer to the character string to be copied from

[ReturnValue]
- Returns a pointer to the character string at the destination of copy.

[Description]
- Copies character string "s2" (ending with NULL) to area "s1"
- After copying, the character string ends with NULL.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strcspn

String Handling Functions

[Function] Calculates the length (number) of unspecified characters that are not found in the other character string

[Format]
```
#include <string.h>

size_t _far strcspn( s1, s2 );
```

[Method] function

[Argument]
const char _far * s1; .... Pointer to the character string to be searched in
const char _far * s2; .... Pointer to the character string to be searched for

[ReturnValue]
- Returns the length (number) of unspecified characters.

[Description]
- Calculates the size of the first character string consisting of characters other than those in 's2' from area 's1', and searches the characters from the beginning of 's1'.
- You cannot search for '\0'.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

stricmp

String Handling Functions

[Function] Compares character strings. (All alphabets are handled as upper-case letters.)

[Format] #include <string.h>

int _far stricmp(s1, s2);

[Method] function

[Argument] char _far * s1; .... Pointer to the first character string to be compared
char _far * s2; .... Pointer to the second character string to be compared

[ReturnValue]
- ReturnValue==0 .... The two character strings are equal.
- ReturnValue>0 .... The first character string (s1) is greater than the other.
- ReturnValue<0 .... The second character string (s2) is greater than the other.

[Description]
- Compares each byte of two character strings ending with NULL. However, all letters are treated as uppercase letters.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strerror

String Handling Functions

[Function] Converts an error number into a character string.

[Format] #include <string.h>

char * _far strerror(errcode);

[Method] function

[Argument] int errcode; .... error code

[ReturnValue]
- Returns a pointer to a message character string for the error code.

[Note]
- stderr returns the pointer for a static array.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strlen

String Handling Functions

[Function] Calculates the number of characters in a character string.

[Format] #include <string.h>

size_t _far strlen(s);

[Method] function

[Argument] const char _far * s; .... Pointer to the character string to be operated on to calculate length

[ReturnValue] ● Returns the length of the character string.

[Description] ● Determines the length of character string "s" (to NULL).

● Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.

strncat

String Handling Functions

[Function] Concatenates character strings ('n' characters).

[Format] #include <string.h>

char _far * _far strncat(s1, s2, n);

[Method] function

[Argument] char _far * s1; .... Pointer to the character string to be concatenated to
const char _far * s2; .... Pointer to the character string to be concatenated from
size_t n; .... Number of characters to be concatenated

[ReturnValue] ● Returns a pointer to the concatenated character string area.

[Description] ● Concatenates character strings "s1" and "n" characters from character string "s2".

● The concatenated string ends with NULL.

● Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.

● When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strncmp

String Handling Function

[Function] Compares character strings ('n' characters).

[Format] #include <string.h>

int _far strncmp(s1, s2, n);

[Method] function

[Argument] const char _far * s1; .... Pointer to the first character string to be compared
const char _far * s2; .... Pointer to the second character string to be compared
size_t n; .... Number of characters to be compared

[ReturnValue]
- ReturnValue==0................The two character strings are equal.
- ReturnValue>0..................The first character string (s1) is greater than the other.
- ReturnValue<0................The second character string (s2) is greater than the other.

[Description]
- Compares each byte of n characters of two character strings ending with NULL.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strncpy

String Handling Function

[Function] Copies a character string ('n' characters).

[Format] #include <string.h>
char _far * _far strncpy(s1, s2, n);

[Method] function

[Argument] char _far * s1; .... Pointer to the character string to be copied to
const char _far * s2; .... Pointer to the character string to be copied from
size_t n; .... Number of characters to be copied

[ReturnValue]
- Returns a pointer to the character string at the destination of copy.

[Description]
- Copies "n" characters from character string "s2" to area "s1". If character string "s2" contains more characters than specified in "n", they are not copied and '\0' is not appended. Conversely, if "s2" contains fewer characters than specified in "n", '\0's are appended to the end of the copied character string to make up the number specified in "n".
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strnicmp

String Handling Functions

[Function] Compares character strings ('n' characters). (All alphabets are handled as uppercase letters.)

[Format] #include <string.h>

int _far strnicmp(s1, s2, n);

[Method] function

[Argument] char _far * s1; .... Pointer to the first character string to be compared
char _far * s2; .... Pointer to the second character string to be compared
size_t n; .... Number of characters to be compared

[ReturnValue]
- ReturnValue==0 .... The two character strings are equal.
- ReturnValue>0 .... The first character string (s1) is greater than the other.
- ReturnValue<0 .... The second character string (s2) is greater than the other.

[Description]
- Compares each byte of n characters of two character strings ending with NULL.However, all letters are treated as uppercase letters.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strpbrk

String Handling Functions

[Function] Searches the specified character in a character string from the other character string.

[Format] #include <string.h>
char _far * _far strpbrk(s1, s2);

[Method] function

[Argument] const char _far * s1; .... Pointer to the character string to be searched in
const char _far * s2; .... Pointer to the character string of the character to be searched for

[ReturnValue]
- Returns the position (pointer) where the specified character is found first.
- Returns NULL if the specified character cannot be found.

[Description]
- Searches the specified character "s2" from the other character string in "s1" area.
- You cannot search for '\0'.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strrchr

String Handling Functions

[Function] Searches the specified character from the end of a character string.

[Format]

```
#include <string.h>

char _far * _far strrchr( s, c );
```

[Method] function

[Argument]
const char _far * s; .... Pointer to the character string to be searched in
int c; .... Character to be searched for

[ReturnValue]
- Returns the position of character "c" that is last encountered in character string "s."
- Returns NULL when character string "s" does not contain character "c".

[Description]
- Searches for the character specified in "c" from the end of area "s".
- You can search for '\0'.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strspn

String Handling Functions

[Function] Calculates the length (number) of unspecified characters that are not found in the other character string.

[Format]

```
#include <string.h>
size_t _far strspn( s1, s2 );
```

[Method] function

[Argument]
const char _far* s1; .... Pointer to the character string to be searched in
const char _far * s2; .... Pointer to the character string of the character to be searched for

[ReturnValue]
- Returns the length (number) of unspecified characters.

[Description]
- Calculates the size of the first character string consisting of characters other than those in 's2' from area 's1', and searches the characters from the beginning of 's1'.
- You cannot search for '\0'.
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strstr

String Handling Functions

[Function] Searches the specified character from a character string.

[Format] #include <string.h>

char _far * _far strstr(s1, s2);

[Method] function

[Argument] const char _far * s1; .... Pointer to the character string to be searched in
const char _far * s2; .... Pointer to the character string of the character to be searched for

[ReturnValue]
- Returns the position (pointer) where the specified character is found.
- Returns NULL when the specified character cannot be found.

[Description]
- Returns the location (pointer) of the first character string "s2" from the beginning of area "s1".
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strtod

Character String Value Convert Functions

[Function] Converts a character string into a double-type integer.

[Format] #include <string.h>

double _far strtod(s,endptr);

[Method] function

[Argument] const char* s; .... Pointer to the converted character string
char ** endptr; .... Pointer to the remaining character strings that have not been converted

[ReturnValue]
- ReturnValue == 0L .... Does not constitute a number.
- ReturnValue != 0L .... Returns the configured number in double type.

[Description]
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strtok

String Handling Functions

[Function] Divides some character string from a character string into tokens.

[Format] #include <string.h>

char _far * _far strtok(s1, s2);

[Method] function

[Argument] char _far * s1; .... Pointer to the character string to be divided up
const char _far * s2; .... Pointer to the punctuation character to be divided with

[ReturnValue]
- Returns the pointer to the divided token when character is found.
- Returns NULL when character cannot be found.

[Description]
- Returns the location (pointer) of the first character string "s2" from the beginning of area "s1".
- In the first call, returns a pointer to the first character of the first token. A NULL character is written after the returned character. In subsequent calls (when "s1" is NULL), this instruction returns each token as it is encountered. NULL is returned when there are no more tokens in "s1".
- Specifying the -DNEAR_LIB* option when compiling selects the high-speed library, which processes the pointer parameters of this function as having the near attribute.
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

* NEAR_LIB specified in the -D option is an identifier for selecting the library from the standard header file string.h.

strtol

Character String Value Convert Function

[Function] Converts a character string into a long-type integer.

[Format] #include <string.h>

long _far strtol(s,endptr,base);

[Method] function

[Argument] const char * s; .... Pointer to the converted character string
char ** endptr; .... Pointer to the remaining character strings that have not been converted.
int base; .... Base of values to be read in (0 to 36)

[ReturnValue]
- ReturnValue == 0L .... Does not constitute a number.
- ReturnValue != 0L .... Returns the configured number in long type.

[Description]
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strtoul

Character String Value Convert Function

[Function] Converts a character string into an unsigned long-type integer.

[Format] #include <string.h>

unsigned long _far strtoul(s,endptr,base);

[Method] function

[Argument] const char* s .... Pointer to the converted character string
char ** endptr; .... Pointer to the remaining character strings that have not been converted.
int base; .... Base of values to be read in (0 to 36)
Reads the format of integral constant if the base of value is zero

[ReturnValue]
- ReturnValue == 0L .... Does not constitute a number.
- ReturnValue != 0L .... Returns the configured number in long type.

[Description]
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

strxfrm

Character String Value Convert Functions

[Function] Converts a character string (using locale information).

[Format]

```
#include    <string.h>

size_t _far strxfrm( s1,s2,n );
```

[Method] function

[Argument]
char* s1; .... Pointer to an area for storing a conversion result character string.
const char* s2; .... Pointer to the character string to be converted.
size_t n; .... Number of bytes converted

[ReturnValue]
- Returns the number of characters converted.

[Description]
- When you specify options -O, -OR, or -OS, the system may selects functions with good code efficiency by optimization.

tan

Mathematical Functions

[Function] Calculates tangent.

[Format]

```
#include    <math.h>

double _far tan( x );
```

[Method] function

[Argument]
double x; .... arbitrary real number

[ReturnValue]
- Returns the tangent of given real number x handled in units of radian.

tanh

Mathematical Functions

[Function] Calculates hyperbolic tangent.

[Format]

```
#include    <math.h>

double _far tanh( x );
```

[Method] function

[Argument] double x; .... arbitrary real number

[ReturnValue] ● Returns the hyperbolic tangent of given real number x.

tolower

Character Handling Functions

[Function] Converts the character from an upper-case to a lower-case.

[Format]

```
#include    <ctype.h>

int tolower( c );
```

[Method] macro

[Argument] int c; .... Character to be converted

[ReturnValue]
- ● Returns the lower-case letter if the argument is an upper-case letter.
- ● Otherwise, returns the passed argument as is.

[Description] ● Converts the character from an upper-case to a lower-case.

toupper

Character Handling Functions

[Function] Converts the character from a lower-case to an upper-case.

[Format]
```
#include     <ctype.h>

int toupper( c );
```

[Method] macro

[Argument] int c; .... Character to be converted

[ReturnValue]
- ● Returns the upper-case letter if the argument is a lower-case letter.
- ● Otherwise, returns the passed argument as is.

[Description]
- ● Converts the character from a lower-case to an upper-case.

ungetc

Input/Output Functions

[Function] Returns one character to the stream

[Format]
```
#include     <stdio.h>

int _far ungetc( c, stream );
```

[Method] macro

[Argument] int c; .... Character to be returned
FILE * stream; .... Pointer of stream

[ReturnValue]
- ● Returns the returned one character if done normally.
- ● Returns EOF if the stream is in write mode, an error or EOF is encountered, or the character to be sent back is EOF.

[Description]
- ● Returns one character to the stream.
- ● Interprets code 0x1A as the end code and ignores any subsequent data.
- ● To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

vfprintf

Input/Output Functions

[Function] Output to a stream with format.

[Format]

```
#include    <stdarg.h>
#include    <stdio.h>

int _far vfprintf( stream,format,ap );
```

[Method] function

[Argument]
FILE * stream; .... Pointer of stream
const char * format; .... Pointer of the format specifying character string
va_list ap; .... Pointer of argument list

[ReturnValue]
- Returns the number of characters output.

[Description]
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

vprintf

Input/Output Functions

[Function] Output to stdout with format.

[Format]

```
#include    <stdarg.h>
#include    <stdio.h>

int _far vprintf( format,ap );
```

[Method] function

[Argument]
const char * format; .... Pointer of the format specifying character string
va_list ap; .... Pointer of argument list

[ReturnValue]
- Returns the number of characters output.

[Description]
- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

vsprintf

Input/Output Functions

[Function] Output to a buffer with format.

[Format]

```
#include     <stdarg.h>
#include     <stdio.h>

int _far vfprintf( s,format,ap );
```

[Method] function

[Argument]

char * s; .... Pointer of the location to be store
const char * format; .... Pointer of the format specifying character string
va_list ap; .... Pointer of argument list

[ReturnValue]

- Returns the number of characters output.

[Description]

- To process the parameter using the far pointer, remake the library file using the make file make.far (makefar.dos in the MS-DOS version).

wcstombs

Multi-byte Character Multi-byte Character String Manipulate Functions

[Function] Converts a wide character string into a multi-byte character string.

[Format]

```
#include     <stdlib.h>

size_t _far wcstombs( s,wcs,n );
```

[Method] function

[Argument]

char * s; .... Pointer to an area for storing conversion multi-byte character string
const wchar_t * wcs; .... Pointer to a wide character string
size_t n; .... Number of wide characters stored

[ReturnValue]

- Returns the number of stored multi-byte characters if the character string was converted correctly.
- Returns -1 if the character string was not converted correctly.

wctomb

Multi-byte Character Multi-byte Character String Manipulate Functions

[Function] Converts a wide character into a multi-byte character.

[Format]
```
#include    <stdlib.h>

int _far wctomb( s,wchar );
```

[Method] function

[Argument]
char * s; .... Pointer to an area for storing conversion multi-byte character string
wchar_t wchar; .... wide character

[ReturnValue]
- Returns the number of bytes contained in the multi-byte characters.
- Returns -1 if there is no corresponding multi-byte character.
- Returns 0 if the wide character is 0.

E.2.4 Using the Standard Library

a. Notes on Regarding Standard Header File

When using functions in the standard library, always be sure to include the specified standard header file. If this header file is not included, the integrity of arguments and return values will be lost, making the program unable to operate normally.

b. Notes on Regarding Optimization of Standard Library

If you specify any of optimization options -O, -OS, or -OR, the system performs optimization for the standard functions. This optimization can be suppressed by specifying -Ono_stdlib. Such suppression of optimization is necessary when you use a user function that bear the same name as one of the standard library functions.

(1)Inline padding of functions

Regarding functions strcpy and memcpy, the system performs inline padding of functions if the conditions in Table E.13 are met.

Table E.13 Optimization Conditions for Standard Library Functions

Function Name	Optimization Condition	Description Example
strcpy	First argument:near pointer Second argument:string constant	strcpy(str, "sample");
memcpy	First argument:near pointer Second argument: string constant Third argument:constant	memcpy(str ,"sample", 6);

(2)Selection of high-speed library (NC30 only)

Some standard library functions have a pointer in argument. NC30 normally handles such pointers as the far pointer. For this reason, NC30 does not generate efficient code if the argument is a near pointer. Therefore, if the argument is a near pointer, the system performs optimization to choose a library function provided for use as near. The table below lists the functions that are subject to such optimization.

Table E.14 Library Functions Subject to Optimization

Function Name	Function Name	Function Name	Function Name
bcopy	strcat	strnicmp	strstr
bzero	strchr	strlen	strspn
memchr	strcmp	strncat	strtod
memcmp	strcoll	strncmp	strtok
memcpy	strcpy	strncpy	strtol
memicmp	strcspn	strnicmp	strtoul
memmove	strerror	strpbrk	strxfrm
memset	stricmp	strrchr	

E.3 Modifying Standard Library

The NC77 package includes a sophisticated function library which includes functions such as the scanf and printf I/O functions. These functions are normally called high-level I/O functions. These high-level I/O functions are combinations of hardware-dependent low-level I/O functions.

In 7700 family application programs, the I/O functions may need to be modified according to the target system's hardware. This is accomplished by modifying the source file for the standard library.

This chapter describes how to modify the NC77 standard library to match the target system.

E.3.1 Structure of I/O Functions

As shown in Figure E.1, the I/O functions work by calling lower-level functions (level 2 ⇒level 3) from the level 1 function. For example, fgets calls level 2 fgetc, and fgetc calls a level 3 function.

Only the lowest level 3 functions are hardware-dependent (I/O port dependent) in the 7700 family. If your application program uses an I/O function, you may need to modify the source files for the level 3 functions to match the system.

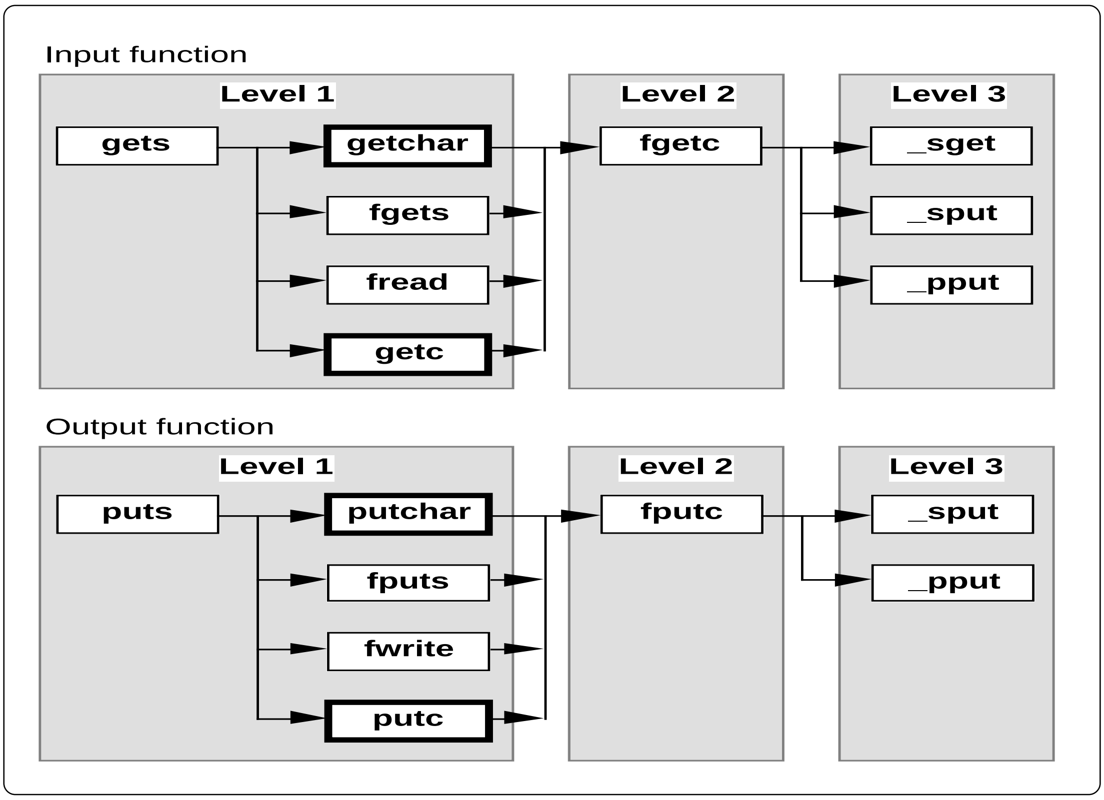


Figure E.1 Calling Relationship of I/O Functions

E.3.2 Sequence of Modifying I/O Functions

Figure E.2 outlines how to modify the I/O functions to match the target system.

a. Modify the level 3 I/O function(s)

↓

b. Set the stream

↓

c. Compile the modified source program(s)

Figure E.2 Example Sequence of Modifying I/O Functions

a. Modifying Level 3 I/O Function

The level 3 I/O functions perform 1-byte I/O via the 7700 family I/O ports. The level 3 I/O functions include _sget and _sput, which perform I/O via the serial communications circuits (UART), and _pput, which performs I/O via the Centronics communications circuit.

[Circuit settings]

- ●Processor mode: Microprocessor mode
- ●Clock frequency: 8MHz
- ●External bus size: 16 bits

[Initial serial communications settings]

- ●Use UART1
- ●Baud rate: 9600bps
- ●Data size: 8 bits
- ●Parity: None
- ●Stop bits: 2 bits

*The initial serial communications settings are made in the init function (init.c).

The level 3 I/O functions are written in the C library source file device.c. Table E.13 lists the specifications of these functions.

Table E.13   Specifications of Level 3 Functions

Input functions	Parameters	Return value (int type)
_sget	None.	If no error occurs, returns the input character Returns EOF if an error occurs
_sput		
_pput		

Output functions	Parameters (int type)	Return value (int type)
_sput	Character to output	If no error occurs, returns 1 Returns EOF if an error occurs
_pput		

Serial communication is set to UART1 in the 7700 family's two UARTs. device.c is written so that the UART0 can be selected using the conditional compile commands, as follows:

●To use UART0 .... #define UART0 1

Specify these commands at the beginning of device.c, or specify following option, when compiling.

●To use UART0 .... -DUART0

To use both UARTs, modify the file as follows:

[1]Delete the conditional compiling commands from the beginning of the device.c file.
[2]Change the UART0 special register name defined in #pragma EQU to a variable other than UART1.
[3]Reproduce the level 3 functions _sget and _sput for UART0 and change them to different variable names such as _sget0 and _sput0.
[4]Also reproduce the speed function for UART0 and change the function name to something like speed0.

This completes modification of device.c.

Next, modify the init function (init.c), which makes the initial I/O function settings, then change the stream settings (see below).

b. Stream Settings

The NC77 standard library has five items of stream data (stdin, stdout, stderr, stdaux, and stdprn) as external structures. These external structures are defined in the standard header file stdio.h and control the mode information of each stream (flag indicating whether input or output stream) and status information (flag indicating error or EOF).

Table E.15 Stream Information

Stream information	Name
stdin	Standard input
stdout	Standard output
stderr	Standard error output (error is output to stdout)
stdaux	Standard auxiliary I/O
stdprn	Standard printer output

The stream corresponding to the NC77 standard library functions shown shaded in Figure E.3 are fixed to standard input (stdin) and standard output (stdout). The stream cannot be changed for these functions. The output direction of stderr is defined as stdout in #define.

The stream can only be changed for functions that specify pointers to the stream as parameters such as fgetc and fputc.

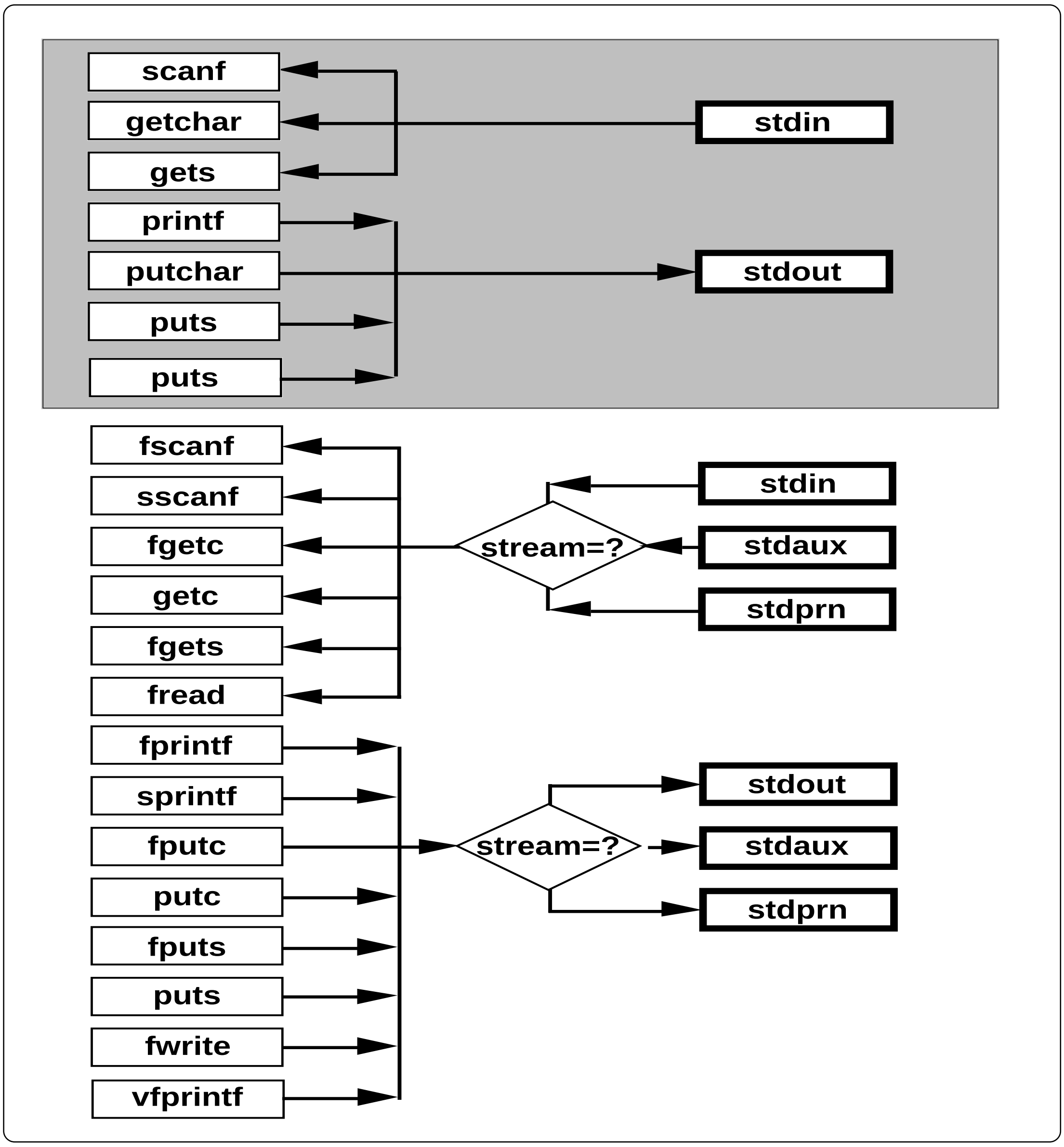


Figure E.3 Relationship of Functions and Streams

Figure E.4 shows the stream definition in stdio.h.

```
/******************************************************************************
*
* standard I/O header file
          :
      (omitted)
          :
typedef struct _iobuf {
        char _buff;             /* Store buffer for ungetc */                   ⇐[1]
        int _cnt;               /* Strings number in _buff(1 or 0) */           ⇐[2]
        int _flag;              /* Flag */                                      ⇐[3]
        int _mod;               /* Mode */                                      ⇐[4]
        int (* _func_in)(void);     /* Pointer to one byte input function */    ⇐[5]
        int (* _func_out)(int);     /* Pointer to one byte output function */   ⇐[6]
} FILE;
#define _IOBUF_DEF
          :
      (omitted)
          :
extern FILE _iob[];
#define  stdin   (&_iob[0])         /* Fundamental input */
#define  stdout  (&_iob[1])         /* Fundamental output */
#define  stdaux  (&_iob[2])         /* Fundamental auxialiary input output */
#define  stdprn  (&_iob[3])         /* Fundamental printer output */

#define  stderr  stdout

/**********************************************************
**********************************************************/
#define _IOREAD 1       /* Read only flag */
#define _IOWRT  2       /* Write only flag */
#define _IOEOF  4       /* End of file flag */
#define _IOERR  8       /* Error flag */
#define _IORW   16      /* Read and write flag */
#define _NFILE  4       /* Stream number */
#define _TEXT   1       /* Text mode flag */
#define _BIN    2       /* Binary mode flag */

      (remainder omitted)
          :
          :
```

Figure E.4 Stream Definition in stdio.h

Let's look at the elements of the file structures shown in Figure E.4. Items [1] to [6] correspond to [1] to [6] in Figure E.4.

[1]char _buff

Functions scanf and fscanf read one character ahead during input. If the character is no use, function ungetc is called and the character is stored in this variable.

If data exists in this variable, the input function uses this data as the input data.

[2]int _cnt

Stores the _buff data count (0 or 1)

[3]int _flag

Stores the read-only flag (_IOREAD), the write-only flag (_IOWRT), the read-write flag (_IORW), the end of file flag (_IOEOF) and the error flag (_IOERR).

- _IOREAD, _IOWRT, _IORW
  These flags specify the stream operating mode. They are set during stream initialization.
- _IOEOF, _IOERR
  These flags are set according to whether an EOF is encountered or error occurs in the I/O function.

[4]int _mod

Stores the flags indicating the text mode (_TEXT) and binary mode (_BIN).

- Text mode
  Echo-back of I/O data and conversion of characters. See the source programs (fgetc.c and fputc.c) of the fgetc and fputc functions for details of echo back and character conversion.
- Binary mode
  No conversion of I/O data. These flags are set in the initialization block of the stream.

[5]int (*_func_in)()

When the stream is in read-only mode (_IOREAD) or read/write mode (_IORW), stores the level 3 input function pointer. Stores a NULL pointer in other cases.

This information is used for indirect calling of level 3 input functions by level 2 input functions.

[6]int (*_func_out)()

When the stream is in write mode (_IOWRT), stores the level 3 output function pointer. If the stream can be input (_IOREAD or _IORW), and is in text mode, it stores the level 3 output function pointer for echo back. Stores a NULL pointer in other cases.

This information is used for indirect calling of level 3 output functions by level 2 output functions.

Set values for all elements other than char_buff in the stream initialization block. The standard library file supplied in the NC77 package initializes the stream in function init, which is called from the ncrt0.a77 startup program.

Figure E.5 shows the source program for the init function.

```
#include <stdio.h>

FILE _iob[4];

void init( void );

void init( void )
{
    stdin->_cnt = stdout->_cnt = stdaux->_cnt = stdprn->_cnt = 0;
    stdin->_flag = _IOREAD;
    stdout->_flag = _IOWRT;
    stdaux->_flag = _IORW;
    stdprn->_flag = _IOWRT;

    stdin->_mod = _TEXT;
    stdout->_mod = _TEXT;
    stdaux->_mod = _BIN;
    stdprn->_mod = _TEXT;

    stdin->_func_in = _sget;
    stdout->_func_in = NULL;
    stdaux->_func_in = _sget;
    stdprn->_func_in = NULL;

    stdin->_func_out = _sput;
    stdout->_func_out = _sput;
    stdaux->_func_out = _sput;
    stdprn->_func_out = _pput;

#ifdef UART0
    speed(_96, _B8, _PN, _S2);
#else
    speed(_96, _B8, _PN, _S2);
#endif
    init_prn();
}
```

Figure E.5   Source file of init function (init.c)

In systems using the two 7700 family UARTs, modify the init function as shown below. In the previous subsection, we set the UART0 functions in the device.c source file temporarily as _sget0, _sput0, and speed0.

[1]Use the standard auxiliary I/O (stdaux) for the UART0 stream.
[2]Set the flag (_flag) and mode (_mod) for standard auxiliary I/O to match the system.
[3]Set the level 3 function pointer for standard auxiliary I/O.
[4]Delete the conditional compile commands for the speed function and change to function speed0 for UART0.

These settings allow both UARTs to be used. However, functions using the standard I/O stream cannot be used for standard auxiliary I/O used by UART0. Therefore, only use functions that take streams as parameters. Figure E.6 shows how to change the init function.

```
void init( void )
{
          :
      (omitted)
          :
    stdaux->_flag = _IORW;          ⇐[2](set read/write mode)
          :
      (omitted)
          :
    stdaux->_mod = _TEXT;           ⇐[2](set text mode)
          :
      (omitted)
          :
    stdaux->_func_in = _sget0;      ⇐[3](set UART0 level 3 input function)
          :
      (omitted)
          :
    stdaux->_func_out = _sput0;     ⇐[3](set UART0 level 3 input function)
          :
      (omitted)
          :
    speed0(_96, _B8, _PN, _S2);     ⇐[4](set UART0 speed function)
    speed(_96, _B8, _PN, _S2);
    init_prn();
}
```

* [2] to [4] correspond to the items in the description of setting, above.

Figure E.6 Modifying the init Function

c. Incorporating the Modified Source Program

There are two methods of incorporating the modified source program in the target system:

[1]Specify the object files of the modified function source files when linking.
[2]Use the makefile (under MS-DOS, makefile.dos) supplied in the NC77 package to update the library file.

In method [1], the functions specified when linking become valid and functions with the same names in the library file are excluded.

Figure E.7 shows method[1]. Figure E.8 shows method[2].

```
% nc77 -c -g -osample  ncrt0.a77 device.r77 init.r77 sample.c<RET>
```

* This example shows the command line when device.c and init.c are modified.

Figure E.7 Method of Directly Linking Modified Source Programs

```
% make <RET>
```

Figure E.8 Method of Updating Library Using Modified Source Programs

Appendix F
Error Messages

This appendix describes the error messages and warning messages output by NC77, and their countermeasures.

F.1 Message Format

If, during processing, NC77 detects an error, it displays an error message on the screen and stops the compiling process.

The following shows the format of error messages and warning messages.

```
nc77:[error-message]
```

Figure F.1 Format of Error Messages from the nc77 Compile Driver

```
[Error(cpp77.error-No.): filename, line-No.] error-message
[Error(ccom): filename, line-No.] error-message
[Fatal(ccom): filename, line-No.] error-message        ←*1
```

Figure F.2 Format of Command Error Messages

```
[Warning(cpp77. warning-No.): filename, line-No.] warning-message
[Warning(ccom): filename, line-No.] warning-message
```

Figure F.3 Format of Command Warning Messages

```
#8 ダブルフォールト    EAAB
CS:EIP = 0248 000S000C     ←*2
```

Figure F.4 Format of DOS-EXTENDER Error Messages (MS-DOS version only)

The following pages list the error messages and their countermeasures. cpp77 messages are listed according to their Nos. The messages output by other programs are listed alphabetically (symbols followed by letters).

*1. Fatal error message
*2. This error message is not normally output. Please contact nearest Mitsubishi office. with details of the message if displayed.

F.2 nc77 Error Messages

Tables F.1 and F.2 list the nc77 compile driver error messages and their countermeasures.

Table F.1 nc77 Error Messages (1/2)

Error message	Description and countermeasure
Arg list too long	● The command line for starting the respective processing system is longer than the character string defined by the system. ⇒Specify a NC77 option to ensure that the number of characters defined by the system is not exceeded. Use the -v option to check the command line used for each processing block.
Cannot analyze error	● This error message is not normally displayed. (It is an internal error.) ⇒Contact Mitsubishi Electric Semiconductor Systems Corp.
Command-file line characters exceed 2048.	● There are more than 2048 characters on one or more lines in the command file. ⇒Reduce the number of characters per line in the command file to 2048 max.
Core dump (*command-name*)	● The processing system (indicated in parentheses) caused a core dump. ⇒The processing system is not running correctly. Check the environment variables and the directory containing the processing system. If the processing system still does not run correctly, Please contact Mitsubishi Electric Semiconductor Systems Corp.
Exec format error	● Corrupted processing system executable file. ⇒Reinstall the processing system.
Ignore option '-?'	● You specified an illegal option (-?) for NC77. ⇒Specify the correct option.
illegal option	● You specified options greater than 100 characters for -rasm77 or -link77. ⇒Reduce the options to 99 characters or less.
Invalid argument	● This error message is not normally displayed. (It is an internal error.) ⇒Contact Mitsubishi Electric Semiconductor Systems Corp.
Invalid option '-?'	● The required parameter was not specified in option "-?". ⇒"-?"Specify the required parameter after "-?".
	● You specified a space between the -? option and its parameter. ⇒Delete the space between the -? option and its parameter.
Invalid option '-o'	● No output filename was specified after the -o option. ⇒Specify the name of the output file. Do not specify the filename extension.

Table F.2 nc77 Error Messages (2/2)

Error message	Description and countermeasure
Invalid suffix '.xxx'	● You specified a filename extension not recognized by NC77 (other than .c, .i, .a77, .r77, .hex). ⇒Specify the filename with the correct extension.
No such file or directory	● The processing system will not run. ⇒Check that the directory of the processing system is correctly set in the environment variable.
Not enough core	[UNIX]: ● Insufficient swap area ⇒Increase the swap area by, for example, adding a secondary swap area. [MS-Windows 95 / NT]: ● Insufficient swap area ⇒Increase the swap area by, for example, adding a secondary swap area. [MS-DOS]: ● Insufficient extended memory ⇒Increase extended memory
Permission denied	● The processing system will not run. ⇒Check access permission to the processing systems. Or, if access permission is OK, check that the directory of the processing system is correctly set in the environment variable.
can't open command file	● Can not open the command file specified by '@'. ⇒ Specify the correct input file.
too many options	● This error message is not normally displayed. (It is an internal error.) ⇒ Contact Mitsubishi Electric Semiconductor Systems Corp.
Result too large	● This error message is not normally displayed. (It is an internal error.) ⇒Contact Mitsubishi Electric Semiconductor Systems Corp.
Too many open files	● This error message is not normally displayed. (It is an internal error.) ⇒Contact Mitsubishi Electric Semiconductor Systems Corp.

F.3 cpp77 Error Messages

Tables F.3 to F.6 list the error messages output by the cpp77 preprocessor and their countermeasures.

Table F.3 cpp77 Error Messages (1/4)

NO.	Error message	Description and countermeasure
1	illegal command option	● Input filename specified twice. ⇒Specify the input filename once only.
		● The same name was specified for both input and output files. ⇒Specify different names for input and output files.
		● Output filename specified twice. ⇒Specify the output filename once only.
		● The command line ends with the -o option. ⇒Specify the name of the output file after the -o option.
		● The -I option specifying the include file path exceeds the limit. ⇒Specify the -I option 8 times or less.
		● The command line ends with the -I option. ⇒Specify the name of an include file after the -I option.
		● The string following the -D option is not of a character type (letter or underscore) that can be used in a macro name. Illegal macro name definition. ⇒Specify the macro name correctly and define the macro correctly.
		● The command line ends with the -D option. ⇒Specify a macro filename after the -D option.
		● The string following the -U option is not of a character type (letter or underscore) that can be used in a macro name. ⇒Define the macro correctly.
		●You specified an illegal option on the cpp77 command line. ⇒Specify only legal options.
11	cannot open input file	● Input file not found. ⇒Specify the correct input file name.
12	cannot close input file	● Input file cannot be closed. ⇒Check the input file name.

Table F.4 cpp77 Error Messages (2/4)

No.	Error message	Description and countermeasure
14	cannot open output file.	● Cannot open output file. ⇒Specify the correct output file name.
15	cannot close output file	● Cannot close output file. ⇒Check the available space on disk.
16	cannot write output file	● Error writing to output file. ⇒Check the available space on disk.
17	input file name buffer over-flow	● The input filename buffer has overflowed. Note that the filename includes the path. ⇒Reduce the length of the filename and path (use the -I option to specify the standard directory).
18	not enough memory for macro identifier	● Insufficient memory for macro name and contents of macro [UNIX]: ⇒Increase the swap area [MS-Windows 95 / NT]: ⇒Increase the swap area [MS-DOS]: ⇒Increase extended memory.
21	include file not found	● The include file could not be opened. ⇒The include files are in the current directory and that specified in the -I option and environment variable. Check these directories.
22	illegal file name error	● Illegal filename. ⇒Specify a correct filename.
23	include file nesting over	● Nesting of include files exceeds the limit (8). ⇒Reduce nesting of include files to a maximum of 8 levels.
25	illegal identifier	● Error in #define. ⇒Code the source file correctly.
26	illegal operation	● Error in preprocess commands #if - #elseif - #as-sert operation expression. ⇒Rewrite operation expression correctly.
27	macro argument error	● Error in number of macro parameters when ex-panding macro. ⇒Check macro definition and reference and correct as necessary.

Table F.5   cpp77 Error Messages (3/4)

No.	Error message	Description and countermeasure
28	input buffer over flow	● Input line buffer overflow occurred when reading source file(s). Or, buffer overflowed when converting macros. ⇒Reduce each line in the source file to a maximum of 1023 characters. If you anticipate macro conversion, modify the code so that no line exceeds 1023 characters after conversion.
29	EOF in comment	● End of file encountered in a comment. ⇒Correct the source file.
31	EOF in preprocess command	● End of file encountered in a preprocess command ⇒Correct the source file.
32	unknown preprocess command	● An unknown preprocess command has been specified. ⇒Only the following preprocess commands can be used in CPP30 : #include, #define, #undef, #if, #ifdef, #ifndef, #else, #endif, #elseif, #line, #assert, #pragma, #error
33	new_line in string	● A new-line code was included in a character constant or character string constant. ⇒Correct the program.
34	string literal out of range 509 characters	● A character string exceeded 509 characters. ⇒Reduce the character string to 509 characters max.
35	macro replace nesting over	● Macro nesting exceeded the limit (20). ⇒Reduce the nesting level to a maximum of 20.
41	include file error	● Error in #include instruction. ⇒Correct.
43	illegal id name	● Error in following macro name or argument in #define command: __FILE__, __LINE__, __DATE__, __TIME__ ⇒Correct the source file.
44	token buffer over flow	● Token character buffer of #define overflowed. ⇒Reduce the number of token characters.
45	illegal undef command usage	● Error in #undef. ⇒Correct the source file.
46	undef id not found	● The following macro names to be undefined in #undef were not defined: __FILE__, __LINE__, __DATE__, __TIME__ ⇒Check the macro name.
52	illegal ifdef / ifndef command usage	● Error in #ifdef. ⇒Correct the source file.

Table F.6   cpp77 Error Messages (4/4)

No.	Error message	Description and countermeasure
53	elseif / else sequence error	● #elseif or #else were used without #if - #ifdef - #ifndef. ⇒Use #elseif or #else only after #if - #ifdef -#ifndef.
54	endif not exist	● No #endif to match #if - #ifdef - #ifndef. ⇒Add #endif to the source file.
55	endif sequence error	● #endif was used without #if - #ifdef - #ifndef. ⇒Use #endif only after #if - #ifdef - #ifndef.
61	illegal line command usage	● Error in #line. ⇒Correct the source file.

F.4 cpp77 Warning Messages

Table F.7 shows the warning messages output by cpp77 and their countermeasures.

Table F.7   cpp77 Warning Messages

No.	Warning Messages	Description and countermeasure
81	reserved id used	● You attempted to define or undefine one of the following macro names reserved by cpp77: __FILE__, __LINE__, __DATE__, __TIME__ ⇒Use a different macro name.
82	assertion warning	● The result of an #assert operation expression was 0. ⇒Check the operation expression.
83	garbage argument	● Characters other than a comment exist after a preprocess command. ⇒Specify characters as a comment (/* *string* */) after the preprocess command.
84	escape sequence out of range for character	● An escape sequence in a character constant or character string constant exceeded 255 characters. ⇒Reduce the escape sequence to within 255 characters.
85	redefined	● A previously defined macro was redefined with different contents. ⇒Check the contents against those in the previous definition.
87	/* within comment	● A comment includes /*. ⇒Do not nest comments.

F.5　nc77 Error Messages

Tables F.8 to F.20 list the nc77 compiler error messages and their countermeasures.

Table F.8　nc77 Error Messages (1/13)

Error message	Description and countermeasure
#pragma PARAMETER *function-name* redefined	● The same function is defined twice in #pragma PARAMETER. ⇒Make sure that #pragma PARAMETER is declared only once.
#pragma PARAMETER & function prototype mismatched	● The function specified by #pragma PARAMETER does not match the contents of argument in prototype declaration. ⇒Make sure it is matched to the argument in prototype declaration.
#pragma PARAMETER's function argument is struct or union	● The struct or union type is specified in the prototype declaration for the function specified by #pragma PARAMETER. ⇒Specify the int or short type, 2-byte pointer type, or enumeration type in the prototype declaration.
#pragma PARAMETER must be declared before use	● A function specified in the #pragma PARAMETER declaration is defined after call for that function. ⇒Declare a function before calling it.
#pragma INTCALL function's argument on stack	● When the body of functions declared in #pragma INTCALL are written in C, the parameters are passed via the stack. ⇒When the body of functions declared in #pragma INTCALL are written in C, specify the parameters are being passed via the stack.
#pragma PARAMETER function's register not allocated	● A register which is specifed in the function decleared by #pragma PARAMETER can not be allocated. ⇒ Use the correct register.
'const' is duplicate	● const is described more than twice. ⇒Write the type qualifier correctly.
'far' & 'near' conflict	● far/near is described more than twice. ⇒Write far/near correctly.
'far' is duplicate	● far is described more than twice. ⇒Write far correctly.
'near' is duplicate	● near is described more than twice. ⇒Write near correctly.
'static' is illegal storage class for argument	● An appropriate storage class is used in argument declaration. ⇒Use the correct storage class.
'volatile' is duplicate	● volatile is described more than twice. ⇒Write the type qualifier correctly.

Table F.9 ccom-mocc Error Messages (2/13)

Error message	Description and countermeasure
(can't read C source from filename line *number* for error message)	● The source line is in error and cannot be displayed. The file indicated by filename cannot be found or the line number does not exist in the file. ⇒Check whether the file actually exists.
(can't open C source filename for error message)	● The source file in error cannot be opened. ⇒Check whether the file exists.
argument type given both places	● Argument declaration in function definition overlaps an argument list separately given. ⇒Choose the argument list or argument declaration for this argument declaration.
array of functions declared	● The array type in array declaration is defined as function. ⇒Specify scalar type struct/union for the array type.
array size is not constant integer	● The number of elements in array declaration is not a constant. ⇒Use a constant to describe the number of elements.
asm()'s string must have 1 $$	● $$ is described more than twice in asm statement. ⇒Make sure that $$ is described only once.
auto variable's size is zero	● An array with 0 elements or no elements was declared in the auto area. ⇒Correct the coding.
bitfield width exceeded	● The bit-field width exceeds the bit width of the data type. ⇒Make sure that the data type bit width declared in the bit-field is not exceeded.
bitfield width is not constant integer	● The bit width of the bit-field is not a constant. ⇒Use a constant to write the bit width.
can't get bitfield address by '&' operator	● The bit-field type is written with the & operator. ⇒Do not use the & operator to write the bit-field type.
can't get inline function's address by '&' operator	● The & operator is written in an inline function. ⇒Do not use the & operator in an inline function.
can't get void value	● An attempt is made to get void-type data as in cases where the right side of an assignment expression is the void type. ⇒Check the data type.
can't output to *file-name*	● The file cannot be wrote ⇒Check the rest of disk capacity or permission of the file.
can't open *file-name*	● The file cannot be opened. ⇒Check the permission of the file.

Table F.10 nc77 Error Messages (3/13)

Error message	Description and countermeasure
can't set argument	● The type of an actual argument does not match prototype declaration. The argument cannot be set in a register (argument). ⇒Correct mismatch of the type.
case value is duplicated	● The value of case is used more than one time. ⇒Make sure that the value of case that you used once is not used again within one switch statement.
conflict declare of *variable-name*	● The variable is defined twice with different storage classes each time. ⇒Use the same storage class to declare a variable twice.
conflict function argument type of *variable-name*	● The argument list contains the same variable name. ⇒Change the variable name.
declared register parameter function's body declared	● The function body for the function declared with #pragma PARAMETER is defined in C ⇒Do not define , in C, the body for such function .
default function argument conflict	● The default value of an argument is declared more than once in prototype declaration. ⇒Make sure that the default value of an argument is declared only once.
default: is duplicated	● The default value is used more than one time. ⇒Use only one default within one switch statement.
do while (struct/union) statement	● The struct or union type is used in the expression of the do-while statement. ⇒Use the scalar type for an expression in the do-while statement.
do while (void) statement	● The void type is used in the expression of the do-while statement. ⇒Use the scalar type for an expression in the do-while statement.
duplicate frame position defind *variable-name*	● ⇒
duplicate 'long'	● long is described more than twice. ⇒Write the type specifier correctly.
Empty declare	● Only storage class and type specifiers are found. ⇒Write a declarator.
float and double not have sign	● Specifiers signed/unsigned are described in float or double. ⇒Write the type specifier correctly.
floating type's bitfield	● A bit-field of an invalid type is declared. ⇒Use the integer type to declare a bit-field.
for (; struct/union;) statement	● The struct or union type is used in the second expression of the for statement. ⇒Use the scalar type to describe the second expression of the for statement.

Table F.11 nc77 Error Messages (4/13)

Error message	Description and countermeasure
for (; void;) statement	● The 2nd expression of the for statement has void. ⇒Use the scalar type as the 2nd expression of the for statement.
function initialized	● An initialize expression is described for function declaration. ⇒Delete the initialize expression.
function member declared	● A member of struct or union is function type ⇒Write the members correctly.
function returning a function declared	● The type of the return value in function declaration is function type. ⇒Change the type to “pointer to function”etc.
function returning an array	● The type of the return value in function declaration is an array type. ⇒Change the type to “pointer to function”etc.
identifier (*variable-name*) is duplicated	● The variable is defined more than one time. ⇒Specify variable definition correctly.
if (struct/union) statement	● The struct or union type is used in the expression of the if statement. ⇒ The expression must have scalar type.
if (void) statement	● The void type is used in the expression of the if statement. ⇒The expression must have scalar type.
illegal storage class for argument, 'inline' ignored	● An inline function is declared in declaration statement within a function. ⇒Declare it outside a function.
illegal storage class for argument, 'interrupt' ignored	● An interrupt function is declared in declaration statement within a function. ⇒Declare it outside a function.
incomplete struct get by []	● An attempt is made to reference or initialize an array of incomplete structs or unions that do not have defined members. ⇒Define complete structs or unions first.
incomplete struct initialized	● An attempt is made to initialize an array of incomplete structs or unions that do not have defined members. ⇒Define complete structs or unions first.
incomplete struct return function call	● An attempt is made to call a function that has as a return value the of incomplete struct or union that does not have defined members. ⇒Define a complete struct or union first.
incomplete struct / union's member access	● An attempt is made to reference members of an incomplete struct or union that do not have defined members. ⇒Define a complete struct or union first.
incomplete struct / union(*tag-name*)'s member access	● An attempt is made to reference members of an incomplete struct or union that do not have defined members. ⇒Define a complete struct or union first.

Table F.12 nc77 Error Messages (5/13)

Error message	Description and countermeasure
inline function's address used	● An attempt is made to reference the address of an inline function. ⇒Do not use the address of an inline function.
inline function's body is not declared previously	● The body of an inline function is not defined. ⇒Using an inline function, define the function body prior to the function call.
invalid '?:' operand	● The ?: operation contains an error. ⇒Check each expression. Also note that the expressions on the left and right sides of : must be of the same type.
invalid '!=' operands	● The != operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '&&' operands	● The && operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '&' operands	● The & operation contains an error. ⇒Check the expression on the right side of the operator.
invalid '&=' operands	● The &= operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '()' operands	● The expression on the left side of ( ) is not a function. ⇒Write a function or a pointer to the function in the left-side expression of ( ).
invalid '*' operands	● If multiplication, the * operation contains an error. If * is the pointer operator, the right-side expression is not pointer type. ⇒For a multiplication, check the expressions on the left and right sides of the operator. For a pointer, check the type of the right-side expression.
invalid '*=' operands	● The *= operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '+' operands	● The + operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '+=' operands	● The += operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '-' operands	● The - operator contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '-=' operands	● The -= operation contains an error. ⇒Check the expressions on the left and right sides of the operator.

Table F.13 nc77 Error Messages (6/13)

Error message	Description and countermeasure
invalid '/=' operands	● The /= operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '<<' operands	● The << operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '<⇐' operands	● The <⇐ operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '⇐' operands	● The ⇐ operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '=' operands	● The = operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '==' operands	● The == operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '>=' operands	● The >= operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '>>' operands	● The >> operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '>>=' operands	● The >>= operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '[ ]' operands	● The left-side expression of [ ] is not array type or pointer type. ⇒Use an array or pointer type to write the left-side expression of [ ].
invalid '^=' operands	● The ^= operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid ' | =' operands	● The | = operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid ' || ' operands	● The || operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid '%=' operands	● The %= operation contains an error. ⇒Check the expressions on the left and right sides of the operator.
invalid ++ operands	● The ++ unary operator or postfix operator contains an error. ⇒ For the unary operator, check the right-side expression. For the postfix operator, check the left-side expression.

Table F.14 nc77 Error Messages (7/13)

Error message	Description and countermeasure
invalid - - operands	● The -- unary operation or postfix operation contains an error. ⇒For the unary operator, check the right-side expression. For the postfix operator, check the left-side expression.
invalid -> used	● The left-side expression of -> is not struct or union. ⇒The left-side expression of -> must have struct or union.
invalid (? :)'s condition	● The ternary operator is erroneously written. ⇒Check the ternary operator.
Invalid #pragma OS Extended function interrupt number	● The INT No. in #pragma OS Extended function is invalid. ⇒Specify correctly.
Invalid #pragma INTCALL interrupt number	● The INT No. in #pragma INTCALL is invalid. ⇒Specify correctly.
Invalid #pragma SPECIAL page number	● The No. in #pragma SPECIAL is invalid. ⇒Specify correctly.
invalid CAST operand	● The cast operation contains an error. The void type cannot be cast to any other type; it can neither be cast from the structure or union type nor can it be cast to the structure or union type. ⇒Write the expression correctly.
invalid asm()'s argument	● The variables that can be used in asm statements are only the auto variable and argument. ⇒Use the auto variable or argument for the statement.
invalid bitfield declare	● The bit-field declaration contains an error. ⇒Write the declaration correctly.
invalid break statements	● The break statement is put where it cannot be used. ⇒Make sure that it is written in switch, while, do-while, and for.
invalid case statements	● The switch statement contains an error. ⇒Write the switch statement correctly.
invalid case value	● The case value contains an error. ⇒Write an integral-type or enumerated-type constant.
invalid cast operator	● Use of the cast operator is illegal. ⇒Write the expression correctly.
invalid continue statements	● The continue statement is put where it cannot be used. ⇒Use it in a while, do-while, and for block.
invalid default statements	● The switch statement contains an error. ⇒Write the switch statement correctly.
invalid enumerator initialized	● The initial value of the enumerator is incorrectly specified by writing a variable name, for example. ⇒Write the initial value of the enumerator correctly.

Table F.15 nc77 Error Messages (8/13)

Error message	Description and countermeasure
invalid function argument	● An argument which is not included in the argument list is declared in argument definition in function definition. ⇒Declare arguments which are included in the argument list.
invalid function's argument declaration	● The argument of the function is erroneously declared. ⇒Write it correctly.
invalid function's default argument	● The default argument of the function is erroneous. ⇒Write it correctly.
invalid function declare	● The function definition contains an error. ⇒Check the line in error or the immediately preceding function definition.
invalid initializer	● The initialization expression contains an error. This error includes excessive parentheses, many initialize expressions, a static variable in the function initialized by an auto variable, or a variable initialized by another variable. ⇒Write the initialization expression correctly.
invalid initializer of *variable-name*	● The initialization expression contains an error. This error includes a bit-field initialize expression described with variables, for example. ⇒Write the initialization expression correctly.
invalid initializer on array	● The initialization expression contains an error. ⇒Check to see if the number of initialize expressions in the parentheses matches the number of array elements and the number of structure members.
invalid initializer on char array	● The initialization expression contains an error. ⇒Check to see if the number of initialize expressions in the parentheses matches the number of array elements and the number of structure members.
invalid initializer on scalar	● The initialization expression contains an error. ⇒Check to see if the number of initialize expressions in the parentheses matches the number of array elements and the number of structure members.
invalid initializer on struct	● The initialization expression contains an error. ⇒Check to see if the number of initialization expressions in the parentheses matches the number of array elements and the number of structure members.
invalid initializer, too many brace	● Too many braces { } are used in a scalar-type initialization expression of the auto storage class. ⇒Reduce the number of braces { } used.

Table F.16 nc77 Error Messages (9/13)

Error message	Description and countermeasure
invalid lvalue	● The left side of the assignment statement is not lvalue. ⇒Write a substitutable expression on the left side of the statement.
invalid lvalue at '=' operator	● The left side of the assignment statement is not lvalue. ⇒Write a substitutable expression on the left side of the statement.
invalid member	● The member reference contains an error. ⇒Write correctly.
invalid member used	● The member reference contains an error. ⇒Write correctly.
invalid redefined type name of (identifier)	● The same identifier is defined more than once in typedef. ⇒Write the identifier correctly.
invalid return type	● The type of return value of the function is incorrect. ⇒ Write it correctly.
invalid sign specifier	● Specifiers signed/unsigned are described twice or more. ⇒Write the type specifier correctly.
invalid storage class for data	● The storage class is erroneously specified. ⇒Write it correctly.
invalid struct or union type	● Structure or union members are referenced for the enumerated type of data. ⇒Write it correctly.
invalid truth expression	● The void, struct, or union type is used in the first expression of a condition expression (?:). ⇒Use scalar type to write this expression.
invalid type specifier	● The same type specifier is described twice or more as in "int int i;" or an incompatible type specifier is described as in "float int i;." ⇒Write the type specifier correctly.
invalid type's bitfield	● A bit-field of an invalid type is declared. ⇒Use the integer type for bit-fields.
invalid unary '!' operands	● Use of the ! unary operator is illegal. ⇒Check the right-side expression of the operator.
invalid unary '+' operands	● Use of the + unary operator is illegal. ⇒Check the right-side expression of the operator.
invalid unary '-' operands	● Use of the - unary operator is illegal. ⇒Check the right-side expression of the operator.
invalid unary '~' operands	● Use of the ~ unary operator is illegal. ⇒Check the right-side expression of the operator.
invalid void type	● The void type specifier is used with “long” or “singed”. ⇒Write the type specifier correctly.

Table F.17 nc77 Error Messages (10/13)

Error message	Description and countermeasure
invalid void type, int assumed	● The void-type variable cannot be declared. Processing will be continued by assuming it to be the int type. ⇒Write the type specifier correctly.
invalid switch statement	● The switch statement is illegal. ⇒Write it correctly.
label *label* redefine	● The same label is defined twice within one function. ⇒Change the name for either of the two labels.
No #pragma ENDASM	● #pragma ASM does not have matching #pragma ENDASM. ⇒Write #pragma ENDASM.
No declarator	● The declaration statement is incomplete. ⇒ Write a complete declaration statement.
Not enough memory	[UNIX version] ● The swap area is insufficient. ⇒Increase the swap area. [MS-Windows 95 / NT version] ● The memory area is insufficient. ⇒Increase the memory or the swap area. [MS-DOS version] ● The extended memory is insufficient. ⇒Increase the extended memory.
not have 'long char'	● Type specifiers long and char are simultaneously used. ⇒Write the type specifier correctly.
not have 'long float'	● Type specifiers long and float are simultaneously used. ⇒Write the type specifier correctly.
not have 'long short'	● Type specifiers long and short are simultaneously used. ⇒Write the type specifier correctly.
not static initializer for *variable-name*	● The initialize expression of static variable contains an error. This is because the initialize expression is a function call, for example. ⇒Write the initialize expression correctly.
not struct or union type	● The left-side expression of -> is not the structure or union type. ⇒Use the structure or union type to describe the left-side expression of ->.
parameter function's body declared	● A function is defined with the same function name that is specified by #pragma PARAMETER. ⇒The function specified by #pragma PARAMETER must be written with the assembly language. Also, if this function has the same name as another assembly-language function, change its name.

Table F.18 nc77 Error Messages (11/13)

Error message	Description and countermeasure
redeclare of *enumerator*	● An enumerator has been declared twice. ⇒Change the name for either of the two enumerators.
redefine function *function-name*	● The function indicated by *function-name* is defined twice. ⇒The function can be defined only once. Change the name for either of the two functions.
redefinition tag of enum *tag-name*	● An enumeration is defined twice. ⇒Make sure that enumeration is defined only once.
redefinition tag of struct *tag-name*	● A structure is defined twice. ⇒Make sure that a structure is defined only once.
redefinition tag of union *tag-name*	● A union is defined twice. ⇒Make sure that a union is defined only once.
reinitialized of *variable-name*	● An initialize expression is specified twice for the same variable. ⇒Specify the initializer only once.
Sorry, stack fram memory exhaust, max. 128 bytes but now *nnn* bytes (NC30, NC308 only)	● A maximum of 128 bytes of parameters can be secured on the stack frame. Currently, nnn bytes have been used. ⇒Reduce the size or number of parameters.
Sorry, stack fram memory exhaust, max. 64(or 255) bytes but now *nnn* bytes	● The stack frame maximum is follows. 64 bytes (NC79) 255bytes (NC30, NC308, NC77 and NC79 with -fDPO8 option used) Currently nnn bytes have been used. ⇒Reduce the auto variables, parameters, and other variables stored in the stack frame area.
Sorry, compilation terminated because of these errors in *function-name*.	● An error occurred in some function indicated by function-name. Compilation is terminated. ⇒Correct the errors detected before this message is output.
Sorry, compilation terminated because of too many errors.	● Errors in the source file exceeded the upper limit (50 errors). ⇒Correct the errors detected before this message is output.
struct or enum's tag used for union	● The tag name for structure and enumerated type is used as a tag name for union. ⇒Change the tag name.
struct or union's tag used for enum	● The tag name for structure and union is used as a tag name for enumerated type. ⇒Change the tag name.
struct or union, enum does not have long or sign	● Type specifiers long or signed are used for the struct/union/enum type specifiers. ⇒ Write the type specifier correctly.
switch's condition is floating	● The float type is used for the expression of a switch statement. ⇒Use the integer type or enumerated type.

Table F.19 nc77 Error Messages (12/13)

Error message	Description and countermeasure
switch's condition is void	● The void type is used for the expression of a switch statement. ⇒Use the integer type or enumerated type.
switch's condition must integer	● Invalid types other than the integer and enumerated types are used for the expression of a switch statement. ⇒Use the integer type or enumerated type.
syntax error	● This is a syntax error. ⇒Write the description correctly.
System Error	● This is an internal error and does not normally occur. ⇒Please contact Mitsubishi Electric Semiconductor Systems Corp.
too many storage class of typedef	● Storage class specifiers such as extern/typedef/static/auto/register are described more than twice in declaration. ⇒Do not describe a storage class specifier more than twice.
type redeclaration of *variable-name*	● The variable is defined with different types each time. ⇒Always use the same type when declaring a variable twice.
typedef initialized	● An initialize expression is described in the variable declared with typedef. ⇒Delete the initialize expression.
undefined label "label" used	● The jump-address label for goto is not defined in the function. ⇒Define the jump-address label in the function.
union or enum's tag used for struct	● The tag name for union and enumerated types is used as a tag name for structure. ⇒Change the tag name.
unknown function argument *variable-name*	● An argument is specified that is not included in the argument list. ⇒Check the argument.
unknown member "*member-name*" used	● A member is referenced that is not registered as any structure or union members. ⇒Check the member name.
unknown pointer to structure identifier "*variable-name*"	● The left-side expression of -> is not the structure or union type. ⇒Use struct or union as the left-side expression of ->.
unknown size of struct or union	● A structure or union is used which has had its size not determined. ⇒Declare the structure or union before declaring a structure or union variable.
unknown structure identifier "*variable-name*"	● The left-side expression of "." dose not have struct or union. ⇒Use the struct or union as it.

Table F.20 nc77 Error Messages (13/13)

Error message	Description and countermeasure
unknown variable "*variable-name*" used in asm()	● An undefined variable name is used in the asm statement. ⇒Define the variable.
unknown variable *variable-name*	● An undefined variable name is used. ⇒Define the variable.
unknown variable *variable-name* used	● An undefined variable name is used. ⇒Define the variable.
void array is invalid type, int array assumed	● An array cannot be declared as void. Processing will be continued, assuming it has type int. ⇒Write the type specifier correctly.
void value can't return	● The value converted to void (by cast) is used as the return from a function. ⇒Write correctly.
while (struct/union) statement	● struct or union is used in the expression of a while statement. ⇒Use scalar type.
while (void) statement	● void is used in the expression of a while statement. ⇒Use scalar type.
multiple #pragma EXT4MPTR's pointer, ignored (NC30 only)	● A pointer variable decleared by #pragma EXT4MPTR is duplecate. ⇒Declare the variable only one time.
zero size array member	● the array which size is zero. ⇒Declare the array size.

F.6 nc77 Warning Messages

Tables F.21 to F.30 list the nc77 compiler warning messages and their countermeasures.

Table F.21 nc77 Warning Messages (1/10)

Warning message	Description and countermeasure
#pragma *pragma-name* & HANDLER both specified	● Both #pragma *pragma-name* and #pragma HANDLER are specified in one function. ⇒ Specify #pragma *pragma-name* and #pragma HANDLER exclusive to each other.
#pragma *pragma-name* & INTERRUPT both specified	● Both #pragma *pragma-name* and #pragma INTERRUPT are specified in one function. ⇒ Specify #pragma *pragma-name* and #pragma INTERRUPT exclusive to each other.
#pragma *pragma-name* & TASK both specified	● Both #pragma *pragma-name* and #pragma TASK are specified in one function. ⇒ Specify #pragma *pragma-name* and #pragma TASK exclusive to each other.
#pragma *pragma-name* format error	● The #pragma *pragma-name* is erroneously written. Processing will be continued. ⇒ Write it correctly.
#pragma *pragma-name* format error, ignored	● The #pragma *pragma-name* is erroneously written. This line will be ignored. ⇒ Write it correctly.
#pragma *pragma-name* not function, ignored	● A name is written in the #pragma *pragma-name* that is not a function. ⇒ Write it with a function name.
#pragma *pragma-name*'s function must be predeclared, ignored	● A function specified in the #pragma *pragma-name* is not declared. ⇒ For functions specified in a #pragma *pragma-name*, write prototype declaration in advance.
#pragma *pragma-name*'s function must be prototyped, ignored	● A function specified in the #pragma *pragma-name* is not prototype declared. ⇒ For functions specified in a #pragma *pragma-name*, write prototype declaration in advance.
#pragma *pragma-name*'s function return type invalid,ignored	● The type of return value for a function specified in the #pragma *pragma-name* is invalid. ⇒ Make sure the type of return value is any type other than struct, union, or double.
#pragma *pragma-name* unknown switch,ignored	● The switch specified in the #pragma *pragma-name* is invalid. ⇒ Write it correctly.

Table F.22 nc77 Warning Messages (2/10)

Warning message	Description and countermeasure
#pragma ADDRESS variable initialized, ADDRESS ignored	● The variable specified in #pragma ADDRESS is initialized. The specification of #pragma ADDRESS will be nullified. ⇒ Delete either #pragma ADDRESS or the initialize expression.
#pragma ASM line too long, then cut	● The line in which #pragma ASM is written exceeds the allowable number of characters = 1,024 bytes. ⇒ Write it within 1,024 bytes.
#pragma directive conflict	● #pragma of different functions is specified for one function. ⇒ Write it correctly.
#pragma DP[n]DATA format error,ignored (NC79 only)	● You have also specified option -fDPO8. ⇒ If you specify both #pragma DP[n]DATA and -fDPO8, #pragma DP[n]DATA is invalid. Delete the option -fDPO8.
	● You have made an error in the format of #pragma DP[n]DATA. ⇒ Correct the format.
#pragma JSRA illegal location, ignored (NC30,NC308 only)	● Do not put #pragma JSRA inside function scope. ⇒ Write #pragma JSRA outside a function.
#pragma JSRW illegal location, ignored (NC30,NC308 only)	● Do not put #pragma JSRW inside function scope. ⇒ Write #pragma JSRW outside a function.
#pragma PARAMETER function's address used	● The address of function specified #pragma PARAMETER is assigned to the pointer variable. ⇒ As don't assign, write correctly.
#pragma control for function duplicate, ignored (NC30,NC308 only)	● Two or more of INTERRUPT, TASK, HANDLER, CYCHANDLER, or ALMHANDLER are specified for the same function in #pragma. ⇒ Be sure to specify only one of INTERRUPT, TASK, HANDLER, CYCHANDLER, or ALMHANDLER.
'auto' is illegal storage class	● An incorrect storage class is used. ⇒ Specify the correct storage class.
'register' is illegal storage class	● An incorrect storage class is used. ⇒ Specify the correct storage class.
-OR, -OS duped option	● Options -OR and -OS are specified simultaneously. ⇒ Specify the option correctly.
argument is define by 'typedef', 'typedef' ignored	● Specifier typedef is used in argument declaration. Specifier typedef will be ignored. ⇒ Delete typedef.
assign far pointer to near pointer, bank value ignored	● The bank address will be nullified when substituting the far pointer for the near pointer. ⇒ Check the data types, near or far.
assignment from const pointer to non-const pointer	● Substitute the constant variable for the non-constant variabel. ⇒ Check the valiable type.

Table F.23 nc77 Warning Messages (3/10)

Warning message	Description and countermeasure
assignment from volatile pointer to non-volatile pointer	● Substitute the volatile variable for the non-volatile variabel. ⇒Check the valiable type.
block level extern variable initialize forbid, ignored	● An initializer is written in extern variable declaration in a function. ⇒Delete the initializer or change the storage class.
can't get address from register storage class variable	● The & operator is written for a variable of the storage class register. ⇒Do not use the & operator to describe a variable of the storage class register.
can't get size of bitfield	● The bit-field is used for the operand of the sizeof operator. ⇒Write the operand correctly.
can't get size of function	● A function name is used for the operand of the sizeof operator. ⇒Write the operand correctly.
can't get size of function, unit size 1 assumed	● The pointer to the function is incremented (++) or decremented (--). Processing will be continued by assuming the increment or decrement value is 1. ⇒Do not increment (++) or decrement (--) the pointer to a function.
char array initialized by wchar_t string	●The array of type char is initialized with type wchar_t . ⇒Make sure that the types of initializer are matched.
case value is out of range	●The value of case exceeds the switch parameter range. ⇒Specify correctly.
character buffer overflow	● The size of the string exceeded 512 characters. ⇒Do not use more than 511 characters for a string.
character constant too long	● There are too many characters in a character constant (characters enclosed with single quotes). ⇒Write it correctly.
constant variable assignment	● In this assign statement, substitution is made for a variable specified by the const qualifier. ⇒Check the declaration part to be substituted for.
cyclic or alarm handler always Bank 0 (NC77,NC79 only)	● Function specified in #pragma CYCHANDLER or ALMHANDLER are always compiled in bank 0 (addresses below 10000H). ⇒None.
cyclic or alarm handler always load DT (NC77,NC79 only)	● There is no need to #pragma LOADDT a function specified in #pragma CYCHANDLER or ALMHANDLER. ⇒Delete #pragma LOADDT.

Table F.24 nc77 Warning Messages (4/10)

Warning message	Description and countermeasure
cyclic or alarm handler function has argument	● The function specified by #pragma CYCHANDLER or ALMHANDLER is using an argument. ⇒The function cannot use an argument. Delete the argument.
enumerator value overflow size of unsigned char	● The enumerator value exceeded 255. ⇒Do not use more than 255 for the enumerator; otherwise, do not specify the startup function -fchar_enumerator.
enumerator value overflow size of unsigned int	● The enumerator value exceeded 65535. ⇒Do not use more than 65535 to describe the enumerator.
enum's bitfield	● An enumeration is used as a bit field member. ⇒Use a different type of member.
external variable initialized, change to public	● An initialization expression is specified for an extern-declared variable. extern will be ignored. ⇒Delete extern.
far pointer (implicitly) casted by near pointer	● The far pointer was converted into the near pointer. ⇒Check the data types, near or far.
function must be far	● The function is declared with the near type. ⇒Write it correctly.
handler function called	● The function specified by #pragma HANDLER is called. ⇒Be careful not to call a handler.
handler function can't return value	● The function specified by #pragma HANDLER is using a returned value. ⇒The function specified by #pragma HANDLER cannot use a returned value. Delete the return value.
handler function has argument	● The function specified by #pragma HANDLER is using an argument. ⇒The function specified by #pragma HANDLER cannot use an argument. Delete the argument.
hex character is out of range	● The hex character in a character constant is excessively long. Also, some character that is not a hex representation is included after \. ⇒Reduce the length of the hex character.
identifier (*member-name*) is duplicated, this declare ignored	● The member name is defined twice or more. This declaration will be ignored. ⇒ Make sure that member names are declared only once.
identifier (*variable-name*) is duplicate	● The variable name is defined twice or more. This declaration will be ignored. ⇒Make sure that variable names are declared only once.

Table F.25 nc77 Warning Messages (5/10)

Warning message	Description and countermeasure
identifier (*variable-name*) is shadowed	● The auto variable which is the same as the name declared as an argument is used. ⇒ Use any name not in use for arguments.
illegal storage class for argument, 'extern' ignored	● An invalid storage class is used in the argument list of function definition. ⇒ Specify the correct storage class.
incompatible pointer types	● The object type pointed to by the pointer is incorrect. ⇒ Check the pointer type.
init elements overflow, ignored	● The initialization expression exceeded the size of the variable to be initialized. ⇒ Make sure that the number of initialize expressions does not exceed the size of the variables to be initialized.
inline function is called as normal function before, change to static function	● The function declared in storage class inline is called as an ordinary function. ⇒ Always be sure to define an inline function before using it.
integer constant is out of range	● The value of the integer constant exceeded the value that can be expressed by unsigned long. ⇒ Use a value that can be expressed by unsigned long to describe the constant.
interrupt function called	● The function specified by #pragma INTERRUPT is called. ⇒ Be careful not to call an interrupt handling function.
interrupt function can't return value	● The interrupt handling function specified by #pragma INTERRUPT is using a return value. ⇒ Return values cannot be used in an interrupt function. Delete the return value.
interrupt function has argument	● The interrupt handling function specified by #pragma INTERRUPT is using an argument. ⇒ Arguments cannot be used in an interrupt function. Delete the argument.
invalid #pragma EQU	● The description of #pragma EQU contains an error. This line will be ignored. ⇒ Write the description correctly.
invalid #pragma SECTION, unknown section base name	● The section name in #pragma SECTION contains an error. The section names that can be specified are data, bss, program, rom, interrupt, and bas. This line will be ignored. ⇒ Write the description correctly.

Table F.26 nc77 Warning Messages (6/10)

Warning message	Description and countermeasure
invalid #pragma operand, ignored	● An operand of #pragma contains an error. This line will be ignored. ⇒Write the description correctly.
invalid function argument	● The expression of the function's argument does not match the type of the function. ⇒ Make sure that the the argument type is matched to the type of the function.
invalid asm's M flag (NC77,NC79 only)	● Error in M flag value in asm statement. ⇒Specify an integer constant (0, 1, or 2).
invalid asm's MX flag, ignored (NC77,NC79 only)	● Error in MX flag value in asm statement. ⇒ Specify an interger constant (0, 1, or 2).
invalid asm's X flag (NC77,NC79 only)	● Error in X flag value in asm statement. ⇒Specify an integer constant (0, 1, or 2).
invalid return type	● The expression of the return statement does not match the type of the function. ⇒Make sure that the return value is matched to the type of the function or that the type of the function is matched to the return value.
invalid storage class for function, change to extern	● An invalid storage class is used in function declaration. It will be handled as extern when processed. ⇒Change the storage class to extern.
Kanji in #pragma ADDRESS	● The line of #pragma ADDRESS contains kanji code. This line will be ignored. ⇒Do not use kanji code in this declaration.
keyword (*keyword*) are reserved for future	● A reversed keyword is used. ⇒Change it to a different name.
mismatch prototyped parameter type	● The argument type is not the type declared in prototype declaration. ⇒Check the argument type.
meaningless statements deleted in optimize phase	● Meaningless statements were deleted during optimization. ⇒Delete meaningless statements.
mismatch function pointer assignment	● The address of a function having a register argument is substituted for a pointer to a function that does not have a register argument (i.e., a non-prototyped function). ⇒Change the declaration of a pointer variable for function to a prototype declaration.
multi-character character constant	● A character constant consisting of two characters or more is used. ⇒Use a wide character (L'xx') when two or more characters are required.
near/far is conflict beyond over typedef	● The type defined by specifying near/far is again defined by specifying near/far when referencing it. ⇒Write the type specifier correctly.

Table F.27 nc77 Warning Messages (7/10)

Warning message	Description and countermeasure
No hex digit no hex digit	● The hex constant contains some character that cannot be used in hex notation. ⇒ Use numerals 0 to 9 and alphabets A to F and a to f to describe hex constants.
No initialized of xxx	● The register argument xxx has been initialized. ⇒ Specify the initializer.
No storage class & data type in declare, global storage class & int type assumed	● The variable is declared without storage-class and type specifiers. It will be handled as int when processed. ⇒ Write the storage-class and type specifiers.
no meaning statement	● A program that has no effect is described. ⇒ None
non-prototyped function used	● A function is called that is not declared of the prototype. This message is output only when you specified the Wnon_prototype option. ⇒ Write prototype declaration. Or delete the option “- Wnon_prototype”.
non-prototyped function declared	● A prototype declaration for the defined function cannot be found. (Displayed only when the -WNP option is specified.) ⇒ Write a prototype declaration.
octal constant is out of range	● The octal constant contains some character that cannot be used in octal notation. ⇒ Use numerals 0 to 7 to describe octal constants.
octal_character is out of range	● The octal constant contains some character that cannot be used in octal notation. ⇒ Use numerals 0 to 7 to describe octal constants.
overflow in floating value converting to integer	● The float value is over the limitaion. ⇒ Discribe the float value inside the range.
old style function declaration	● Decleare the function by K&R style. ⇒ Decleare the function by ANSI style.
prototype function is defined as non-prototype function before.	● The non-prototyped function is redefine prototype-declaration. ⇒ Unite ways to declare function type.
redefined type name of (xxx)	● The same typedef is defined twice. ⇒ Make sure that typedef is defined only once.
register parameter function used before as stack parameter function	● The function for register argument is used as a function for stack argument before. ⇒ Write a prototype declaration before using the function.
section name is renamed twice	● The section name of a data section is changed twice or more using #pragma SECTION. ⇒ Make sure that the section of a data section is changed only once.

Table F.28 nc77 Warning Messages (8/10)

Warning message	Description and countermeasure
sorry, get stack's address, but DT not 0 (NC77,NC79 only)	● This error occurs when the -bank option is specified. When the address of an auto variable is assigned to a pointer and an object referenced using that pointer, DT points to outside bank 0, preventing bank 0 from being referenced. ⇒ Declare the variable as a far type.
size of incomplete type	● An undefined structure or union is used in the operand of the sizeof operator. ⇒ Define the structure or union first.
size of incomplete type	● The number of elements of an array defined as an operand of the sizeof operator is unknown. ⇒ Define the structure or union first.
size of void	● Get the size of void type variable by sizeof operation. ⇒ Discribe collectly
static valuable in inline function	● static data is declared within a function that is declared in storage class inline. ⇒ Do not declare static data in an inline function.
string size bigger than array size	● The size of the initialize expression is greater than that of the variable to be initialized. ⇒ Make sure that the size of the initialize expression is equal to or smaller than the variable.
string terminator not added	● Since the variable to be initialized and the size of the initialize expression are equal, '\0' cannot be affixed to the character string. ⇒ Increase a element number of array.
struct (or union) member's address can't has no near far information	● near or far is used as arrangement position information of members (variables) of a struct (or union). ⇒ Do not specify near and far for members.
task function called	● The function specified by #pragma TASK is called. ⇒ Be careful not to call a task function.
task function can't return value	● The function specified by #pragma TASK is using a return value. ⇒ The function specified by #pragma TASK cannot use return values. Delete the return value.
task function has invalid argument	● Argument for the task start function is invalid. ⇒ You can only write void or int type. Correct as necessary.
this comparison is always false	● Comparison is made that always results in false. ⇒ Check the conditional expression.
this comparison is always true	● Comparison is made that always results in true. ⇒ Check the conditional expression.

Table F.29 nc77 Warning Messages (9/10)

Warning message	Description and countermeasure
this feature not supported now, ignored	● This is a syntax error. Do not this syntax because it is reserved for extended use in the future. ⇒Write the description correctly.
this function used before with non-default argument	● A function once used is declared as a function that has a default argument. ⇒Declare the default argument before using a function.
this interrupt function is called as normal function before	● A function once used is declared in #pragma INTERRUPT. ⇒An interrupt function cannot be called. Check the content of #pragma.
too big octal character	● The character constant or the octal constant in the character string exceeded the limit value (255 in decimal). ⇒Do not use a value greater than 255 to describe the constant.
too few parameters	● Arguments are insufficient compared to the number of arguments declared in prototype declaration. ⇒Check the number of arguments.
too many parameters	● Arguments are excessive compared to the number of arguments declared in prototype declaration. ⇒Check the number of arguments.
uncomplete array access	● An incomplete multidimensional array has been accessed. ⇒Specify the size of the multidimensional array.
unknown #pragma STRUCT xxx	● #pragma STRUCTxxx cannot be processed. This line will be ignored. ⇒Write correctly.
unknown debug option (-dx)	● The option -dx cannot be specified. ⇒Specify the option correctly.
unknown function option (-Wxxx)	● The option -Wxxx cannot be specified. ⇒Specify the option correctly.
unknown function option (-fx)	● The option -fx cannot be specified. ⇒Specify the option correctly.
unknown function option (-gx)	● The option -gx cannot be specified. ⇒Specify the option correctly.
unknown optimize option (-mx)	● The option -mx cannot be specified. ⇒Specify the option correctly.
unknown optimize option (-Ox)	● The option -Ox cannot be specified. ⇒Specify the option correctly.
unknown option (-x)	● The option -x cannot be specified. ⇒Specify the option correctly.

Table F.30 nc77 Warning Messages (10/10)

Warning message	Description and countermeasure
unknown pragma pragma-specification used	● Unsupported #pragma is written. ⇒Check the content of #pragma. *This warning is displayed only when the -Wunknown_pragma (-WUP) option is specified.
wchar_t array initialized by char string	● The initialize expression of the wchar_t type is initialized by a character string of the char type. ⇒Make sure that the types of the initialize expression are matched.
zero divide in constant folding	● The divisor in the divide operator or remainder calculation operator is 0. ⇒Use any value other than 0 for the divisor.
zero divide, ignored	● The divisor in the divide operator or remainder calculation operator is 0. ⇒Use any value other than 0 for the divisor.
zero width for bitfield	● The bit-field width is 0. ⇒Write a bit-field equal to or greater than 1.
assignment in comparison statement	● You put an assignment expression in a comparison statement. ⇒You may confuse "==" with '='. Check on it.
meaningless statement	● The tail of a statement is "==". ⇒You may confuse "=" with '=='. Check on it.

Appendix G
The Stack Size Calculation Utility (stk77)

This appendix describes how to start the stack size calculation utility stk77 and its command line options.

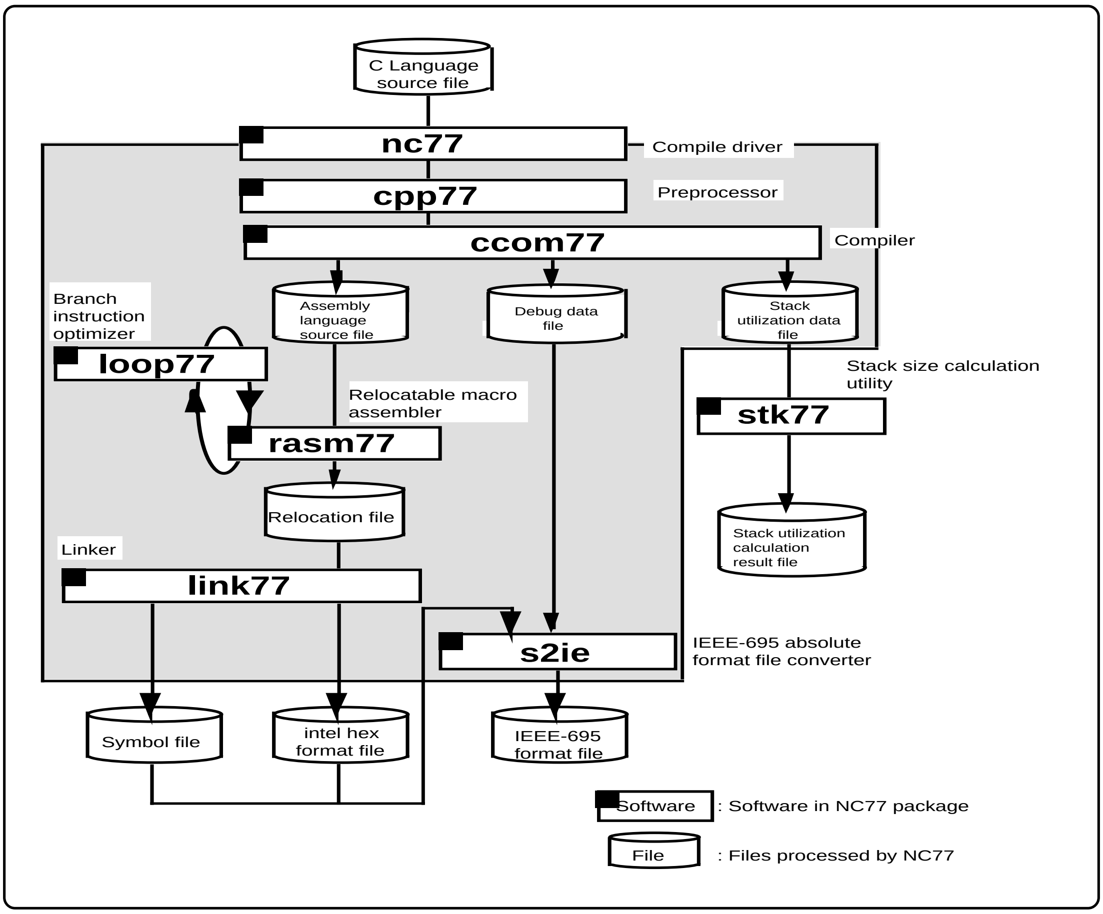
G.1 Introduction of stk77

G.1.1 Introduction of stk77 processes

The stk77 stack size calculation utility processes the stack utilization display file (extension .stk) generated when the -fshow_stack_usage (-fSSU) command line option is specified with the nc77 compile driver. It calculates the stack size required for the program to run and the function call relationship (C flow). The following information is required in order to run stk77.

1. Stack utilization display file (mandatory)
2. Symbol file*1 (optional)
3. Command line option(s) (optional)

Figure G.1 illustrates the NC77 processing flow.



FigureG.1 NC77 Processing Flow

*1.If you specify the symbol file (extension .sym) as an stk77 option, you do not need to specify the stack utilization display file for the respective source file. When you specify a symbol file, the stack utilization display files corresponding to all source files for that symbol file are required.

G.1.2 Stack Utilization Display File

The stack utilization display file is output when the nc77 command line option -fshow_stack_usage (-fSSU) is specified when compiling. The file extension is .stk. The stk77 stack size calculation utility bundled with NC77 calculates the stack size used by specified individual files from the stack utilization display file.

```
FUNCTION main                                  ⇐[1]
    context        5 bytes                     ⇐[2]
    auto           2 bytes                     ⇐[3]
    f8regSize      0 bytes                     ⇐[4]
           6 bytes PUSH &   CALL printf        ⇐[5]
           6 bytes PUSH (MAX)                  ⇐[6]
============================================================
```

Figure G.2   Example Stack Utilization Display File

The contents of the stack utilization display file are shown below. Items [1] to [6] correspond to [1] to [6] in Figure G.2.

[1]Shows the name of the function
[2]Shows the return address stored in the stack when that function is called, or the size used for the old frame pointer (DPR)
[3]Shows the stack size used for the storage class auto or as a temporary area
[4]No. of bytes for internal register for 64-bit floating point operations
[5]Shows the number of bytes pushed to the stack when the function is called, and the function name
[6]Shows the maximum number of bytes pushed by the function

Parameters pushed when the function is called are calculated as part of the stack size on the calling side. It is not possible to identify if functions were called indirectly by pointers (indicated as 0 byte PUSH & CALL (indirect call)).

G.2 Starting stk77

G.2.1 stk77 Command Line Format

For starting stk77, you have to specify the information and parameter that required.

● Direct specification of stack utilization display file

% stk77Δ[*command-line-option*]Δ<*stack-utilization-display-file-name*>

● Specifying stack utilization display file from map file

% stk77Δ[*command-line-option*]Δ-m<*map-file-name*>

% :Prompt
< > :Mandatory item
[ ] :Optional item
Δ :Space
Delimit multiple command line options with spaces.

Figure G.3 stk77 Command Line Format

The following information file is required in order to run stk77.

1. Stack utilization display file (mandatory)
2. Symbol file (optional)

The following nc77 command line options are specified:

- Output of stack utilization display file (extension .stk): .... -fSSU
- Output of relocatable object file (extension .r77): .... -c

The following stk77 options are also specified:

- Output of calculation result display file (extension .siz): .... -o
- Calculation start function: .... -e
- File indicating the amount of stacks used for library functions .... -l

```
%nc77 -c -fSSU sample.c<RET>
NC77 COMPILER for 7700 FAMILY V.5.10 Release 1
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
and MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

sample.c


%
%stk77 -etimer_a0int -o sample.stk -lLINDEFLT.stk<RET>
NC77 STACK UTILITY stk77 for 7700 V.1.10.01
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

*** Stack Size ***

          292 bytes

%
```

<RET> : Means entering the return key.

*1 The calculation start function name of stk77 is timer_a0int.

*2 The name of a file indicating the amount of stacks used for library functions bears is nc77lib.stk.

Figure G.4 Example stk77 Command Line

G.2.2 stk77 Command Line Options

The following information(input parameters) is needed in order to start stk77.

Table G.1 shows the stk77 command line options.

Table G.1 stk77 Command Line Options

Option	Description
-e<*function name*>	Specifies the function name started calculating stack size. If this option is omitted ,starts calculating stuck size from the main function.
-s<*symbol file name*>	Specify the symbol file name.
-o	Output stuck size and a display of function call relations to the calculation result display file (extension .siz).
-c	Output a display of function call relations to standard output device of the host machine (EWS or PC).
-l<*file name*>	Specifies the file name, corresponding to a library file, which indicates the amount of stacks used.

-e *function name*

Specify function

Function : Specifies the name of the function at which to start calculation of stack utilization. If this option is omitted, the stack size is calculated starting with the 'main' function.

Syntax : stk77Δ-e*function name*Δ[*command line option*]Δ<*name of stack utilization display file*>

Execution example :

```
% stk77-efunc1 sample.stk
NC77 STACK UTILITY stk77 for 7700 V.1.10.XX
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

*** Stack Size ***

            514 bytes

%
```

Notes : To specify a stack utilization display file for a source file that does not include the main function, you must specify the name of the first function.

-s *symbol file name*

Specify map file

Function : Specifies the name of the symbol file that includes the source file for which the stack size is to be calculated. If you specify the symbol file, you do not need to specify the stack utilization display file.

Syntax : stk77Δ[*command line option*]Δ-m<*name of map file*>

Execution example :

```
% stk77 -ssample.sym
NC77 STACK UTILITY stk77 for 7700 V.1.10.XX
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

*** Stack Size ***

            514 bytes

%
```

Notes :
1. Only one symbol file can be specified.
2. If specifying the option -s, create a symbol file by specifying the linker's startup option "-link77Δ-s" when compiling by nc77.

-o

Generate calculation result display file

Function : Outputs the stack size and function call relationship to the calculation result display file (extension .siz)

Execution example :

```
% stk77 -o func1 sample.stk
NC77 STACK UTILITY stk77 for 7700 V.1.10.XX
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

*** Stack Size ***

            514 bytes

%
```

-c

Display function call relationship

Function : Outputs the function call relationship to the host machine's (engineering work-station or personal computer) standard output

Execution example :

```
%stk77 -c sample.stk
NC77 STACK UTILITY stk77 for 7700 V.1.10.XX
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

*** Stack Size ***

           514 bytes

*** C Flow ***

main(sample.stk)
        func1(sample.stk)
                func2(sample.stk)
%
```

-l *stack utilization display file name for library functions*

Specifying a stack usage display file for library functions

Function : Specify a stack utilization display file for library functions.

Syntax : stk77Δ[*command-line-option*]Δ-l<*stack-utilization-display-file-name*>

Execution example :

```
% stk77 -lnc77lib.stk sample.stk
NC77 STACK UTILITY stk77 for 7700 V.1.10.XX
Copyright 1999 MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

*** Stack Size ***

            514 bytes

%
```

Notes :

1. Before using functions included in the library file (i.e., when specifying a library file using the -l option when compiling by nc77), be sure to create a stack utilization display file for library functions first.

2. NC77 comes with a file for calculating the amount of stacks used that is provided for the library file (nc77lib.lib). When calculating the amount of stacks used for the library file (nc77lib.lib) you use, specify "- nc77lib.stk" with the -l option.

G.3 Controlling Relationship Between stk77 Function Calls

When calculating stack sizes, stk77 cannot calculate the stack size of such function calls as shown in Table G.2. If the program includes such function calls, the messages shown in Table G.2 are output to the screen and to the calculation result display file. In such cases, the indicated stack size is the maximum value that can be calculated (XXbytes in Table G.2).

Table G.2   Function Call Relationship and Messages

Function call relationship	Message
Recursive calls in program	XX bytes + *'function name'
Indirect call in program	XX bytes + Indirect Call
No data in input file on functions that make up the program ❋ The message shows 0 bytes + 'function name' if there is no 'main' function or function specified in the -e option.	XX bytes + 'function name'

NC77 cannot generate a stack utilization display file for assembler functions. Therefore, if the program includes assembler functions, calculate the required stack sizes separately and then create the stack utilization display file.

Also, stk77 cannot calculate the amount of stack used by functions called using the asm function.

G.4 Example of stk77 use

G.4.1 Calculating User Stack Section Size

Stack utilization can be determined by processing the stack utilization display file using the stk77 stack size calculation utility. Figure G.5 shows an example stk77 command line, while Figure G.6 is an example of the calculation result display file.

```
% stk77 -o smp.siz -lnc77lib.stk<RET>
```

% : Prompt
smp.stk : Name of stack utilization display file
-lnc77lib.stk : The stack utilization display file,nc77lib.stk is specified by option "-l".

Figure G.5 Example stk77 Command Line

```
*** Stack Size ***

           84 bytes        ⇐Shows the stack size to be used.


*** C Flow ***

main(smp.stk)              ⇐Shows function that calls C function(s).
        func1(smp.stk)
        func2(smp.stk)
                _i4mod(C:/lib77/nc77lib.stk)
                _i4div(C:/lib77/nc77lib.stk)
```

Figure G.6 Calculation Result Display File (smp.siz)

Sets the stack size calculating above. Figure G.7 is an example of setting the stack size.

```
;----------------------------------------------------------------------
; STACK SIZE definition
;----------------------------------------------------------------------
STACKSIZE       .equ    54h
```

Figure G.7 Example of Setting the User Stack Size

G.4.2 Calculating the Stack Size to use interrupt functions

Usually, the stk77 recursively tracks respective functions by using the "main" functin as a base point and calculates the maximum stack size. Thus you need to separately obtain the stack size for use with interrupt functions, indirect calling function, and the like.

Here follows the way of obtaining the stack size for use with interrupt functions.

```
#pragma INTERRUPT func3    /* Declaration that func3 is interrupt function */

void   main();
int   func1(  int,   int);
int   func2(  int,   long,   int);
void   func3(void);
int    func4(int);

int s = 0;
int ss = 0;

void main()                /* function main */
{
      int    i, j, k;        /* auto valuable 6 Byte (used) */

    k = func1(i, k);
    k = func2(i, j, k);
}
                :
                :
            (omitted)
                :
                :

void func3( void )              /* interrupt function func3 */
{
        s = func4(ss);
}

int func4(a)            /*function func4 */
int a;
{
        a++;
        return a;
}
```

Figure G.8 C language sample program (smp2.c)

a. Calculating stack utilization using stk77

The stk77 stack size calculation utility can calculates from any function. The interrupt function described in the sample program shows Figure G.8 is func3. Therefore, calculate the amount of stacks used from func3. For calculate the amount of stacks used from func3, specifies func3 using stk77 command line option '-e'. Figure G.9 shows an example stk77 command line, while Figure G.10 is an example of the calculation result display file.

```
%stk77 -o -efunc3 smp2.stk
```

%   :Prompt
smp2.stk   :Name of stack utilization display file

Figure G.9   An example stk77 command line

```
** Stack Size ***

           23 bytes


*** C Flow ***

func3(smp2.stk)
        func4(smp2.stk)
```

Figure G.10   The calculation result display file (smp2.stk)

Note) Using multiple interrupt, add the stack size of the function for multiple interrupt.

G.5 stk77 Error Messages

G.5.1 Error Messages

Table G.3 lists the stk77 stack size calculation utility error messages and their countermeasures.

Table G.3 stk77 Error Messages

Error message	Contents of error and corrective action
Usage : stk77 [option...] filename...<ret>	● The command input format is incorrect. ⇒Check the command input format, then reinput.
not enough memory	● The host machine's available memory is insufficient. ⇒Increase the capacity of available memory by deleting unnecessary drivers, etc.
target file not found	● The corresponding file cannot be found. ⇒Check whether your specified file exists.
invalid file format	● The file format is incorrect. ⇒Check whether the file format is correct.

G.5.2 Warning Messages

Table G.4 lists the stk77 stack size calculation utility warning messages and their countermeasures.

Table G.4 stk77 Warning Messages

Warning Message	Contents of warning and corrective action
cannot open 'file name'.	● The indicated file cannot be opened. ⇒Check the file.
cannot close 'file name'.	● The indicated file cannot be closed. ⇒Check the file.
invalid option 'xxx'	● Option is erroneously specified. ⇒Input options correctly.
Ignore option 'xxx'.	● An option is specified cannot be used in stk77. ⇒Input a correct option.

Appendix H
IEEE-695 Object Format Converter (s2ie)

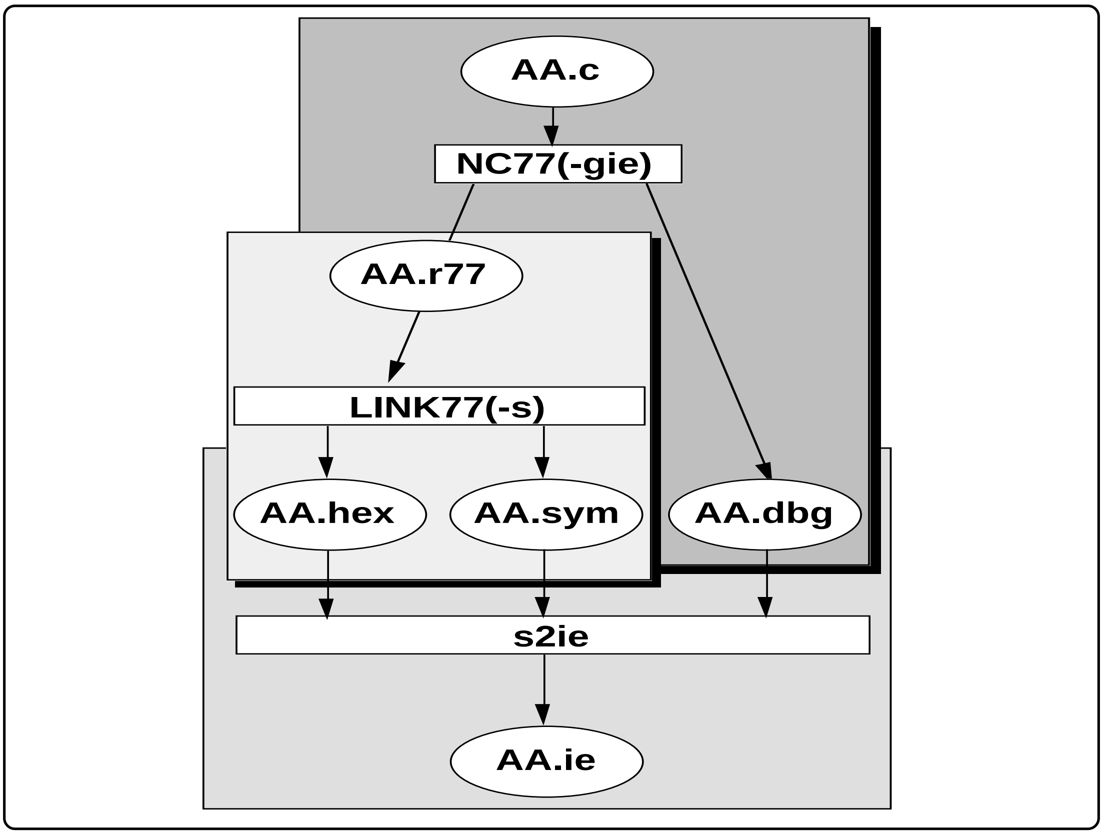
This appendix describes how to start the IEEE-695 Object Fromat Converter s2ie and its command line options.

H.1 Introduction of s2ie

The IEEE-695 absolute-format file converter s2ie puts together the files given below to generate a debugging information file (having the extension .ie) in IEEE-695 format. An IEEE-695 absolute-format file is required for using the Mitsubishi-supplied debugger and simulator-debugger to reference C language information such as an auto variable, a structure, and the like.

- Files that the e2ie puts together
  1. The C-language debugging information file (this file is generated if you select the compilation option -gie, and is given the extension .dbg.)
  2. The symbol file (this file is generated if you select the linkage option -s, and is given the extension .sym.)
  3. The hexadecimal machine-language file (having the extension .hex.)

Selecting the compilation option -gie causes the s2ie to automatically start up from the compiler. [*1]



FigureH.1 s2ie Processing Flow

*1. The s2ie is started up after link77 operates at the final stage of compilation. Thus if you stop the operation by use of one of the options, -E, -P, -S, and -c, then the s2ie is not started up.

H.2 Starting s2ie

H.2.1 s2ie Command Line Format

For starting s2ie, you have to specify the information and parameter that required.

% s2ieΔ[*command-line-option*]Δ<*symbol-file-name*>

% :Prompt
< > :Mandatory item
[ ] :Optional item
Δ :Space
Delimit multiple command line options with spaces.

Figure G.3 stk77 Command Line Format

H.2.2 s2ie Command Line Options

Table H.1 shows the s2ie command line options.

Table H.1 s2ie Command Line Options

Option	Description
-.	Suppresses the copyright message display at startup.
-NLS	Outputs a file in absolute IEEE-695 format (having the extension .ie), but doesn't output local symbols contained in the assembly language file to the IEEE-695 file.
-V	Display the startup message of the s2ie programs, then finishes processing(without conversion).
-o<*file name*>	Specifies the name of the genarated by s2ie.
-Wstdout	Outputs error message to the host machine's standard output(stdout).

H.3 Notes

The s2ie puts together the C-language debugging information file that the compiler generates and the symbol file that the linker generates. On this account, there can be instances in which consistency between the C-language debugging information and symbol information cannot be maintained if you carry out partial re-compilation by use of a make program or the like.

Errors that occur in the course of working the s2ie are probably due to the above-mentioned inconsistency, so compile all the source files again and link them again.

H.4 Example of s2ie use

H.4.1 s2ie controled by compile drive

The s2ie is automatically started up by the compilation option -gie.

When the compilation is over, a file having the same root name as the hexadecimal machine-language file and the extension .ie is generated.

```
% s2ie77 -gie ncrt0.a77 sample.c<RET>
          :
      (omitted)
          :
%ls
    ncrt0.a77         ncrt0.hex         section.inc       test.dbg
    ncrt0.ie          ncrt0.sym         test.c
```

Figure H.3 Example s2ie controled by compile drive(nc77)

H.4.2 using s2ie directly

To use a make program or the like, you work the compiler and the linker on an individual basis, so you need to directly start up the s2ie.

```
%nc77 -c -g ncrt0.a77<RET>
          :
      (omitted)
          :
%nc77 -c -gie sample.c<RET>
          :
      (omitted)
          :
%link77 ncrt0.r77 test.r77, , ,-s<RET>
          :
      (omitted)
          :
%s2ie ncrt0.sym
```

Figure H.4 exsample using s2ie directly

H.5 s2ie Error Messages

H.5.1 Error Messages

Table H.3 lists the s2ie IEEE-695 absolute format file converter error messages and their countermeasures.

Table H.3 s2ie Error Messages(1/2)

No.	Error message	Contents of error and corrective action
100	Ignore symbol filename 'filename'	● Incorrrect filename. ⇒Check the filename.
101	Can't open symbol file 'filename'	● Specified file does not exist. ⇒Check the file.
102	Can't read symbol file 'filename'	● Cannot read the specified file. ⇒Check the file.
103	Can't seek symbol file 'filename'	● Cannot seek the specified file. ⇒Check the file.
104	Can't malloc	● Cannot allocate memory. ⇒Increase avalable memory
105	The file has no data 'filename'	● There is no data in the specified file ⇒Check the file.
200	Illegal symbol file format.	● Incorrect symbol file format. ⇒Regenerate the symbol file.
201	Illegal symbol file SECTION format.	
202	Illegal symbol file FUNCTION format.	
203	Illegal symbol file LOCAL LABEL format.	
204	Illegal symbol file GLOBL LABEL format.	
205	Illegal symbol file SOURCE format.	
206	Illegal symbol file LANGUAGE format.	
207	I found ._end before ._func.	● Incorrect debug file format. ⇒Recompile.
208	I found new SCOPE out of functions.	
209	I found unknown type 'function type'	
210	Illegal index number 'index No.'	
211	I don't know this type 'type No.'	
212	Illegal function variable 'variable type'	
300	Can't ope IEEE file	● Cannot generate IEEE format file. ⇒Check the file.
301	Can't write IEEE file	● Cannot write IEEE format file ⇒Check the file.
302	Can't seek IEEE file	● Cannot seek IEEE format file. ⇒Check the file.

Table H.3 s2ie Error Messages(2/2)

No.	Error message	Contents of error and corrective action
303	Can't open hex file	● No hex format file. ⇒Check the file.
304	Illegal hex address	● Cannot coordinate with symbol file. ⇒Recompile.
305	No section	● There is no symbol data. ⇒Recompile.

H.5.2 Warning Messages

Table H.4 lists the s2ie IEEE-695 absolute format file converter warning messages and their countermeasures.

Table H.4 s2ie Warning Messages

No.	Warning Message	Contents of warning and corrective action
500	Can't file address 'symbol name'	● Cannot determine the address corresponding to a symbol name. ⇒Recompile.
501	Can't open .dbg file 'symbol name'	● There is no debug data file. ⇒Recompile

Technical Support Communication Sheet

Date :___/___/___(Total Pages)

To Distributor:

A text file the installer generates in the following directory can be used instead of this sheet.

\SUPPORT\Product-name\SUPPORT.TXT

Contact Address	Product Information
Company :	Product name :
Department :	Version :
Responsible person :	License ID :
Phone :	- - - -
FAX :	Host Machine :
E-mail :	OS: Ver.
Address :	
Message :	

If this form does not have sufficient space, use another sheet of paper to write your information.

NC77 V.5.20 User's Manual

First Edition: November 1, 1999
Document No.: MSD-NC77-UE-991101

Renesas Technology Corp.

NC77 V.5.20
User's Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan