PREFACE

■ Objective of This Manual and Target Readers

This manual describes the SOFTUNE C compiler (hereinafter referred to as the C compiler) usage procedures and libraries.

This manual is prepared for persons who use the above-mentioned compiler and create and develop application programs in C language.

This manual is to be read by persons who have a basic knowledge of each MCU (Micro Controller Unit).

The compiler described in this manual conforms to the American National Standard for Information Systems — Programming Language C, X3.159-1989, which is abbreviated "ANSI standard" in this manual.

■ Notes on Trademarks

SOFTUNE is trademark of Fujitsu Ltd.

F2MC, an abbreviation of Fujitsu Flexible Microcontroller, is the registered trademark of Fujitsu Ltd.

Microsoft, Windows, Windows NT, and MS-DOS are registered trademarks of Microsoft Corp. in the USA and/or other countries.

UNIX is a registered trademark that X/Open Co., Ltd. has licensed in the United States and other countries.

■ Composition of Manual

This manual consists of the following 10 chapters and appendix.

Chapter 1 OVERVIEW OF SOFTUNE C COMPILER

This chapter outlines the C compiler.

Chapter 2 SETUP OF SYSTEM ENVIRONMENT BEFORE USING C COMPILER

This chapter describes the C compiler operating environment variables.

Chapter 3 OPERATION

This chapter describes the command function specifications.

Chapter 4 OBJECT PROGRAM STRUCTURE

This chapter describes the information necessary for program execution.

Chapter 5 EXTENDED LANGUAGE SPECIFICATIONS

This chapter describes the extended language specifications supported by the compiler and the limitations on compiler translation.

Chapter 6 EXECUTION ENVIRONMENT

This chapter describes the user program execution procedure to be performed in an environment where no operating system exists.

Chapter 7 LIBRARY OVERVIEW

This chapter outlines the C libraries by describing the organization of files provided by the libraries and the relationship to the system into which the libraries are incorporated.
Chapter 8  LIBRARY INCORPORATION
This chapter describes the processes and functions to be prepared for library use.

Chapter 9  COMPILER-DEPENDENT SPECIFICATIONS
This chapter describes the specifications that vary with the compiler.
The description is related to the JIS standard based on the ANSI standard.

Chapter 10  SIMULATOR DEBUGGER LOW-LEVEL FUNCTION LIBRARY
This chapter describes how to use the simulator debugger low-level function library.

APPENDIX
The appendix gives a list of types, macros, and functions provided by the library and the operations specific to the libraries (A,B). Notes when F^2MC-16LX CPU is used are described (C). The guide to change the function call interface to the register argument passing is described (D). The list of the error message is described (E).

■ Syntax Books
For C language syntax and standard library functions, refer to commercially available ANSI standard compliant reference books.

■ Reference Books
• The C Programming Language
  (Brian W. Kernighan & Dennis M. Ritchie)
• Japanese edition entitled Programming Language C UNIX Type Programming Method and Procedure
  (Translated by Haruhisa Ishida; Kyoritsu Shuppan)
• American National Standard for Information Systems - Programming Language C, X3.159-1989
  (Western Electric Company, Incorporated)
• UNIX System User's Manual System V
  (Western Electric Company, Incorporated)
• UNIX System V Programmer Reference Manual
  (AT&T Bell Laboratories)
• User Reference Manual UTS/5 Release 0.1
  (Western Electric Company, Incorporated and Amdahl Corporation)
• UTS Command Reference Manual UTS/5 Release 0.1
  (Western Electric Company, Incorporated and Amdahl Corporation)
• Japanese Industrial Standards Programming Language C
  (Japan Standards Association)
• The contents of this document are subject to change without notice. Customers are advised to consult with FUJITSU sales representatives before ordering.
• The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of Fujitsu semiconductor device; Fujitsu does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. Fujitsu assumes no liability for any damages whatsoever arising out of the use of the information.
• Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of Fujitsu or any third party or does Fujitsu warrant non-infringement of any third-party’s intellectual property right or other right by using such information. Fujitsu assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.
• The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).
• Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.
• If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the prior authorization by Japanese government will be required for export of those products from Japan.
# CONTENTS

## CHAPTER 1 OVERVIEW OF SOFTUNE C COMPILER
- 1.1 C COMPILER FUNCTIONS .................................................................2
- 1.2 BASIC PROCESS OF COMMANDS .....................................................3
- 1.3 C COMPILER BASIC FUNCTIONS .........................................................5

## CHAPTER 2 SETUP OF SYSTEM ENVIRONMENT BEFORE USING C COMPILER
- 2.1 FETOOL .........................................................................................8
- 2.2 OPT907 .......................................................................................9
- 2.3 INC907 .....................................................................................10
- 2.4 TMP ..........................................................................................11
- 2.5 FELANG .....................................................................................12

## CHAPTER 3 OPERATION
- 3.1 COMMAND LINE ........................................................................14
- 3.2 COMMAND OPERANDS .................................................................15
- 3.3 FILE NAMES AND DIRECTORY NAMES ............................................16
- 3.4 COMMAND OPTIONS .....................................................................17
  - 3.4.1 List of Command Options .........................................................18
  - 3.4.2 List of Command Cancel Options ............................................20
- 3.5 DETAILS OF OPTIONS .................................................................22
  - 3.5.1 Translation Control Related Options .........................................23
  - 3.5.2 Preprocessor Related Options ..................................................25
  - 3.5.3 Data Output Related Options ...................................................28
  - 3.5.4 Language Specification Related Options .................................33
  - 3.5.5 Optimization Related Options ..................................................37
  - 3.5.6 Output Object Related Options ...............................................44
  - 3.5.7 Debug Information Related Options .........................................49
  - 3.5.8 Command Related Options .......................................................50
  - 3.5.9 Option File Related Options .....................................................51
- 3.6 OPTION FILES .............................................................................52
- 3.7 MESSAGES GENERATED IN TRANSLATION PROCESS ...................54

## CHAPTER 4 OBJECT PROGRAM STRUCTURE
- 4.1 SECTION STRUCTURE .................................................................58
- 4.2 MEMORY MODELS .......................................................................61
- 4.3 GENERATION RULES FOR NAMES USED BY COMPILER ................63
- 4.4 BOUNDARY ALIGNMENT .............................................................64
- 4.5 BIT FIELD ..................................................................................65
- 4.6 STRUCTURE/UNION .................................................................67
- 4.7 FUNCTION CALL INTERFACE (STACK ARGUMENT PASSING) ............69
  - 4.7.1 Stack Frame (stack argument passing) ........................................70
  - 4.7.2 Argument (stack argument passing) ...........................................72
  - 4.7.3 Argument Extension Format (stack argument passing) ...............73
  - 4.7.4 Calling Procedure (stack argument passing) ..............................74
  - 4.7.5 Register (stack argument passing) ............................................75
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Extended Language Specifications</td>
<td>91</td>
</tr>
<tr>
<td>5.1</td>
<td>Assembler Description Functions</td>
<td>92</td>
</tr>
<tr>
<td>5.2</td>
<td>Interrupt Control Functions</td>
<td>95</td>
</tr>
<tr>
<td>5.3</td>
<td>I/O Area Access Function</td>
<td>97</td>
</tr>
<tr>
<td>5.4</td>
<td>Direct Area Access Function</td>
<td>98</td>
</tr>
<tr>
<td>5.5</td>
<td>16-Bit/24-Bit Addressing Access Function</td>
<td>100</td>
</tr>
<tr>
<td>5.6</td>
<td>In-Line Expansion Specifying Function</td>
<td>102</td>
</tr>
<tr>
<td>5.7</td>
<td>Section Name Change Function</td>
<td>103</td>
</tr>
<tr>
<td>5.8</td>
<td>Register Bank Number Setup Function</td>
<td>106</td>
</tr>
<tr>
<td>5.9</td>
<td>Interrupt Level Setup Function</td>
<td>107</td>
</tr>
<tr>
<td>5.10</td>
<td>System Stack Use Specifying Function</td>
<td>108</td>
</tr>
<tr>
<td>5.11</td>
<td>Stack Bank Automatic Distinction Function</td>
<td>109</td>
</tr>
<tr>
<td>5.12</td>
<td>No-Register-Save Interrupt Func. Function</td>
<td>110</td>
</tr>
<tr>
<td>5.13</td>
<td>Built-In Function</td>
<td>111</td>
</tr>
<tr>
<td>5.14</td>
<td>Predefined Macros</td>
<td>116</td>
</tr>
<tr>
<td>5.15</td>
<td>Limitations on Compiler Translation</td>
<td>117</td>
</tr>
<tr>
<td>6</td>
<td>Execution Environment</td>
<td>119</td>
</tr>
<tr>
<td>6.1</td>
<td>Execution Process Overview</td>
<td>120</td>
</tr>
<tr>
<td>6.2</td>
<td>Startup Routine Creation</td>
<td>122</td>
</tr>
<tr>
<td>7</td>
<td>Library Overview</td>
<td>125</td>
</tr>
<tr>
<td>7.1</td>
<td>File Organization</td>
<td>126</td>
</tr>
<tr>
<td>7.2</td>
<td>Relationship to Library Incorporating System</td>
<td>129</td>
</tr>
<tr>
<td>8</td>
<td>Library Incorporation</td>
<td>131</td>
</tr>
<tr>
<td>8.1</td>
<td>Library Incorporation Overview</td>
<td>132</td>
</tr>
<tr>
<td>8.2</td>
<td>Initialization/Termination Process Required for Library Use</td>
<td>133</td>
</tr>
<tr>
<td>8.3</td>
<td>Low-Level Function Types</td>
<td>134</td>
</tr>
<tr>
<td>8.4</td>
<td>Standard Library Functions and Required Process/Low-Level Functions</td>
<td>135</td>
</tr>
<tr>
<td>8.5</td>
<td>Low-Level Function Specifications</td>
<td>136</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Open Function</td>
<td>137</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Close Function</td>
<td>138</td>
</tr>
<tr>
<td>8.5.3</td>
<td>Read Function</td>
<td>139</td>
</tr>
<tr>
<td>8.5.4</td>
<td>Write Function</td>
<td>140</td>
</tr>
<tr>
<td>8.5.5</td>
<td>Seek Function</td>
<td>141</td>
</tr>
</tbody>
</table>
CHAPTER 1  OVERVIEW OF SOFTUNE C COMPILER

This chapter outlines the C compiler. The C compiler is a language processor program which translates source programs written in C language into the assembly language for Fujitsu-provided various microcontroller units.

1.1  C COMPILER FUNCTIONS
1.2  BASIC PROCESS OF COMMANDS
1.3  C COMPILER BASIC FUNCTIONS
1.1 C COMPILER FUNCTIONS

When a C source file expressed in the C language is described, the C compiler generates an assembler source file which is expressed in assembly language.

C Compiler Functions

The processing steps for assembler source file generation are indicated below.

- Preprocessing
  
  Preprocessing is conducted by the preprocessor (cpps) which is a subcomponent of the compiler. Preprocessing instructions (#if, #define, #include, etc.) in a C source file are interpreted and converted to a preprocessed C source file.

- Compilation
  
  Compilation is conducted by the compiler (ccoms). The preprocessed C source file is converted to an assembler source file.

For the use of the C compiler, the fcc907s command is to be used. This command automatically calls up the tools composing the C compiler (preprocessor and compiler), and provides control over C source file compiling. The C compiler structure is shown in Figure 1.1-1.

![C Compiler Structure Diagram]

Figure 1.1-1 C Compiler Structure

In the subsequent sections, the C compiler translation process is explained using commands. For the details of the command function specifications, see CHAPTER 3, “OPERATION”.
1.2 BASIC PROCESS OF COMMANDS

The basic process of commands used by the C compiler is described below.

- **fcc907s:** For F²MC-16 family

**fcc907s Command Basic Process**

The fcc907s command basically generates an object file from an described C source file. The command regards any file with a .c extension as a C source file.

An example of using the fcc907s command is given below, where “>” is the command prompt.

**[Example 1]**

> fcc907s -cpu MB90F553A file.c

At the input given above, the command regards file.c as a C source file and, if no error is detected, generates an object file (file.obj) in the current directory.

**[Example 2]**

> fcc907s -o outfile -cpu MB90F553A file.c

At the input given above, the command generates an object file (outfile). The command operation process can be controlled by specifying options, such as -o.

**Options for Compiling Process Control**

- **-P option**
  When the -P option is specified, the command calls up the preprocessor only and performs preprocessing to generate a preprocessed C source file in the current directory. The extension of the generated file is changed to .i.

- **-S option**
  When the -S option is specified, the command calls up the preprocessor and compiler and performs preprocessing and compiling to generate an assembler source file in the current directory. The extension of the generated file is changed to .asm.

- **-o option**
  When the -o option is specified, the command generates the file specified in the command line as a result of processing.

Output files generated according to the above options specifying can be used as the input files for the fcc907s command. The input files and output files generated by options are shown in Figure 1.2-1.
Figure 1.2-1 Relationship between Input Files and Output Files Generated by Options
1.3 C COMPILER BASIC FUNCTIONS

The C compiler functions are described below.

- **Header file search**
- **Coordination with symbolic debugger**
- **Optimization**

The symbolic debugger is a support tool for analyzing a program created in C language.

### Header File Search

The header file can be acquired using the C program #include instruction. When the absolute path name is specified, the header file enclosed within angular brackets (<> ) is searched for in the directory specified by absolute path name. When the absolute path name is not specified, the standard directory is searched.

The standard header file is supplied by the C compiler.

The header file enclosed by double quotation marks (" ) is searched for in a directory specified by the absolute path name. If the absolute path name is not specified, such a header file is searched for in a directory having a file containing a #include line. If the header file is not found in a directory having a file containing a #include line, the standard directory is searched next.

The -I option makes it possible to add a directory for header file search.

[Example]

```
> fcc907s -cpu MB90F553A -I ..\include file.c
```

At the input given above, the command searches for the header file enclosed within angular brackets in the order shown below.

1. ..\include
2. Standard directory

The header file enclosed by double quotation marks is searched for in the order shown below.

1. Current directory having a file containing a #include line
2. ..\include
3. Standard directory

The -I option can be specified a desired number of times. When it is specified two or more times, search operations are conducted in the specified order.
CHAPTER 1 OVERVIEW OF SOFTUNE C COMPILER

■ Coordination with Symbolic Debugger

When the -g option is specified, the compiler generates the debug information to be used by the symbolic debugger. When such information is generated, C language level debugging can be accomplished within the symbolic debugger. Two types of symbol debuggers are available; simulator debugger and emulator debugger.

When the optimization option (-O[1-4]) is specified, the compiler attempts to ensure good code generation by changing the computation target position and eliminating computations that are judged to be unnecessary. To minimize the amount of data exchange with memory, the compiler tries to retain data within a register. It is therefore conceivable that a break point positioned in a certain line may fail to cause a break or that currently monitored certain address data may fail to vary with the expected timing. It also well to remember that the debug data will not be generated for an unused local variable or a local variable whose area need not be positioned in a stack as a result of optimization.

Debugging must be conducted with the above considerations taken into account.

■ Optimization

When the -O option is specified, the compiler generates an object subjected to general-purpose optimization.
This chapter describes the C compiler operating environment variables (for the setting of environment variables, refer to the manual for each operating system). All the environment variables can be omitted.

For the supply style, refer to the *C Compiler Installation Manual*. The Windows version permits the use of long file names for the directories to be set up as environment variables. For the characters applicable to long file names, see 3.3, “FILE NAMES AND DIRECTORY NAMES”.

[Setup Example]

```
set FETOOL=c:Fujitsu MCU tool
```

For directory name specified by environment variable, do not use double quotation marks (".

2.1 FETOOL
2.2 OPT907
2.3 INC907
2.4 TMP
2.5 FELANG
CHAPTER 2 SETUP OF SYSTEM ENVIRONMENT BEFORE USING C COMPILER

2.1 FETOOL

Specify the installation directory for the development environment.

FETOOL

[General Format 1] For UNIX OS
   setenv FETOOL Installation directory

[General Format 2] For Windows
   set FETOOL=Installation directory

The command accesses the compiler, message file, include file, and other items via the path specified by FETOOL.

When FETOOL setup is not completed, the parent directory for the directory where the activated command exists (the .. position of the directory where the command exists) is regarded as the installation directory.

No more than one directory can be specified.

[Example] For UNIX OS
   setenv FETOOL /usr/local/softune

[Example] For Windows
   set FETOOL=c:\softune
2.2 OPT907

Specify the directory for the default option file to be used by the command.

- OPT907
  - [General Format 1] For UNIX OS
    setenv OPT907 Default option file directory
  - [General Format 2] For Windows
    set OPT907=Default option file directory

Specify the directory for the default option file to be used by the command.
If OPT907 setup is not completed, the directory placed at a relativity from the directory specified by FETOOL (>FETOOL\lib\907) is regarded as the default option file directory.

No more than one directory can be specified.

- [Example] For UNIX OS
  setenv OPT907 /usr/local/softune/lib\907

- [Example] For Windows
  set OPT907=c:\softune\lib\907
2.3 INC907

Specify the directory where a standard header file search is to be conducted by the command.

**INC907**

[General Format 1] For UNIX OS
setenv INC907 Standard include directory

[General Format 2] For Windows
set INC907=Standard include directory

Specify the directory where the standard header file is to be searched for. The directory specified by INC907 is regarded as the standard include directory.

If INC907C setup is not completed, the directory placed at a relativity from the directory specified by FETOOL (>FETOOL>\lib\907\include) is regarded as the standard header file directory.

No more than one directory can be specified.

[Example] For UNIX OS
setenv INC907 /usr/local/softune/lib/907/include

[Example] For Windows
set INC907=c:\softune\lib\907\include
2.4 TMP

Specify the directory for the temporary file to be used by the C compiler.

TMP

[General Format 1] For UNIX OS
   setenv TMP Temporary directory

[General Format 2] For Windows
   set TMP=Temporary directory

Specify the working directory for creating the temporary file to be used by the C compiler.

If TMP setup is not completed, the temporary file is created in the \tmp directory for UNIX OS or in the current directory for Windows.

No more than one directory can be specified.

[Example] For UNIX OS
   setenv TMP /usr/tmp

[Example] For Windows
   set TMP=c:\tmp
FELANG

Specify the code for messages.

- FELANG

  [General Format 1] For UNIX OS
  
  setenv FELANG Message code

  [General Format 2] For Windows

  set FELANG=Message code

Specify the message code. The following codes can be specified.

- ASCII: Outputs messages in ASCII code
  The generated messages are in English.
  Select this code for a system without a Japanese language environment.

- EUC: Outputs messages in EUC code
  The generated messages are in Japanese.

- SJIS: Outputs messages in SHIFT JIS code
  The generated messages are in Japanese.

If FELANG setup is not completed, the ASCII code is considered to be selected.

[Example] For UNIX OS

  setenv FELANG EUC

[Example] For Windows

  set FELANG=SJIS
CHAPTER 3  OPERATION

This chapter describes the command function specifications.

3.1 COMMAND LINE
3.2 COMMAND OPERANDS
3.3 FILE NAMES AND DIRECTORY NAMES
3.4 COMMAND OPTIONS
3.5 DETAILS OF OPTIONS
3.6 OPTION FILES
3.7 MESSAGES GENERATED IN TRANSLATION PROCESS
CHAPTER 3 OPERATION

3.1 COMMAND LINE

The command line format is shown below.

- fcc907s [options] operands

Command Line

Options and operands can be specified in the command line. They can be specified at any position within the command line. Two or more options and operands can be specified. Options can be omitted.

Option and operand entries are to be delimited by a blank character string. The command recognizes the options and operands in the order shown below.

1. An entry beginning with a hyphen (-) is first recognized as an option. The subsequent character string is interpreted to determine the option type.
2. As regards an option having an argument, the subsequent character string is regarded as the argument.
3. The remaining entries in the command line are recognized as operands.

[Example]

>fcc907s file1.c file2.c -S -I \home\myincs -cpu MB90F553A

At first, -S and -I are regarded as options. Since the -I option has an argument, the subsequent character string \home\myincs is regarded as the argument. The remaining entries (file1.c and file2.c) are regarded as operands.

Options : -S, -I \home\myincs
Operands : file1.c, file2.c

Command Process

The command calls up the preprocessor, compiler, assembler, and linker for all input files in the order of their specifying and performs preprocessing, compiling, assembling, and linking. The results are output into files which are named by replacing the input file extensions with .obj.

[Example]

>fcc907s file1.c file2.c file3.c -cpu MB90F553A

Files named file1.c, file2.c, and file3.c are subjected to preprocessing, compiling, and assembling so that files named file1.obj, file2.obj, and file3.obj are generated.
3.2 COMMAND OPERANDS

One or more input files can be specified as operands.

■ Command Operands

The command determines the file type according to the input file extension and performs an appropriate process to suit the file type.

The extension cannot be omitted.

- File Specifying
  C source files, preprocessed C source files, assembler source files, and object files can be specified as operands.

- File Extension
  The relationship between input file extensions and command processes is shown in Table 3.2-1.

Table 3.2-1 Relationship between Extensions and Command Processes

<table>
<thead>
<tr>
<th>Extension</th>
<th>Command Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>.c</td>
<td>The file having this extension is regarded as a C source file and subjected to preprocessing and subsequent processes.</td>
</tr>
<tr>
<td>.i</td>
<td>The file having this extension is regarded as a preprocessed C source file and subjected to compiling and subsequent processes.</td>
</tr>
<tr>
<td>.asm</td>
<td>The file having this extension is regarded as a compiled assembler source file and subjected to assembling and subsequent processes.</td>
</tr>
<tr>
<td>.obj</td>
<td>For this type of file, the command does nothing.</td>
</tr>
<tr>
<td>.abs</td>
<td>The file having this extension is regarded as a linked absolute file, and an error output is generated. No absolute file can be specified.</td>
</tr>
</tbody>
</table>

Note: however, that the associated process may be inhibited depending on the option specifying.

[Example]

> fcc907s file1.c file2.i -cpu MB90F553A

A file named file1.c is subjected to preprocessing, compiling, and assembling. A file named file2.i is then subjected to compiling and assembling to generate files named file1.obj and file2.obj.
3.3 FILE NAMES AND DIRECTORY NAMES

The following characters are applicable to file names and directory names.

- **File Names and Directory Names**
  - **Windows version**
    Alphanumeric characters, symbols except \, /, :, ", ?, <, >, and \, Shift-JIS kanji codes, and Shift-JIS 1-byte kana codes.
    When long file name is specified as option and operand, it should be enclosed by double quotation marks ("'). However, do not use double quotation marks at setup environment variable with this file name.
  - **Other Versions**
    Underbar (_) and alphanumeric characters (however, the first character must be the underbar or alphabetical character).
  - **Module Name**
    The module name is based on a file name. It is formed by an underbar (_) and alphanumeric characters (The first character must be underbar or alphabetical character). If other characters are used for the file name, the characters that cannot be used for the module name are converted to underbars. File names allowing identical module names after conversion should not be used.
3.4 COMMAND OPTIONS

This section describes the command options.

■ Option Syntax

The option consists of a hyphen (-) and one or more characters following the hyphen. Some options have an argument. A blank character string must be positioned between an option and an argument. The command options cannot be grouped for purposes of specifying. Grouping is a technique of specifying which, for instance, uses a -Sg form to specify both the -S option and -g option.

■ Multiple Specifying of Same Option

If the same option is specified more than one time, only the last-specified option in the command line is assumed to be valid.

[Example]

```bash
>fcc907s -o outfile file.c -o outobj -cpu MB90F553A
```

The resultant output file name will be outobj. The following options can be specified more than one time, and they are significant by each specifications.

- Options that are significant when specified more than one time
  -D  -f  -I  -INF  -K  -T  -U  -x  -Y

When the above options are specified more than one time, see details of options.

■ Position within Command Line

The option's position within the command line does not have a special meaning. Options are interpreted in the same manner no matter where in the command line they are specified.

[Example]

1) >fcc907s -C -E file1.c file2.c -cpu MB90F553A
2) >fcc907s file1.c -E file2.c -C -cpu MB90F553A

The same processing operations are performed for cases 1) and 2).

■ Exclusiveness and Dependency

Some options are mutually exclusive or dependent on each other. For option exclusiveness and dependency, see details of options.

■ Case Sensitivity

As regards the options, their upper-case and lower-case characters are different from each other. For example, the -O option is different from the -o option. However, the upper- and lower-case characters of suboptions are not differentiated from each other. For example, the -K eopt option is considered in the same as the -K EOPT option. The suboptions are the character strings that follow the -K option or -INF option.
3.4.1 List of Command Options

When executed without argument specifying, the command outputs an option list to the standard output. The options for the command are listed in Table 3.4-1 and Table 3.4-2. The options listed in the tables can be recognized by the command.

### List of Command Options

#### Table 3.4-1 List of Command Options

<table>
<thead>
<tr>
<th>Specifying Format</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-B</code></td>
<td>Allows the C++ style comments(//)</td>
</tr>
<tr>
<td><code>-C</code></td>
<td>Leaves a comment in the preprocessing result</td>
</tr>
<tr>
<td><code>-cmsg</code></td>
<td>Outputs the compiling process end message to the standard output</td>
</tr>
<tr>
<td><code>-cpu MB number</code></td>
<td>Specifies the MB number of the CPU to be used</td>
</tr>
<tr>
<td><code>-cwno</code></td>
<td>Sets end code to 1 when warning given</td>
</tr>
<tr>
<td><code>-D name[=tokens]</code></td>
<td>Defines the macro name</td>
</tr>
<tr>
<td><code>-E</code></td>
<td>Performs preprocessing only and outputs the result to the standard output</td>
</tr>
<tr>
<td><code>-f filename</code></td>
<td>Specifies the option file</td>
</tr>
<tr>
<td><code>-g</code></td>
<td>Adds the information necessary for debugging to the object</td>
</tr>
<tr>
<td><code>-H</code></td>
<td>Outputs the acquired header file pathname to the standard output</td>
</tr>
<tr>
<td><code>-help</code></td>
<td>Outputs the option list to the standard output</td>
</tr>
<tr>
<td><code>-I dir</code></td>
<td>Specifies the directory for head file search</td>
</tr>
<tr>
<td><code>-INF LIST</code></td>
<td>Generates the assemble list</td>
</tr>
<tr>
<td>`-INF [SRCIN</td>
<td>LINENO]`</td>
</tr>
<tr>
<td><code>-INF STACK[filename]</code></td>
<td>Generates the stack use amount data</td>
</tr>
<tr>
<td>`-J {a</td>
<td>c}`</td>
</tr>
<tr>
<td>`-K {DCONST</td>
<td>FCONST}`</td>
</tr>
<tr>
<td><code>-K EOPT</code></td>
<td>Effects optimization for changing the arithmetic operation evaluation procedure</td>
</tr>
<tr>
<td><code>-K LIB</code></td>
<td>Recognizes the standard function operation and implements in-line expansion/substitution for other functions</td>
</tr>
<tr>
<td><code>-K NOALIAS</code></td>
<td>Effects optimization on the presumption that different pointers do not indicate the same area</td>
</tr>
<tr>
<td><code>-K NOINLIB</code></td>
<td>Effects no in-line expansion for interrupt related functions</td>
</tr>
<tr>
<td><code>-K NOUNROLL</code></td>
<td>Inhibits loop unrolling</td>
</tr>
<tr>
<td><code>-K NOVOLATILE</code></td>
<td>Does not consider __io qualifier variables to be volatile</td>
</tr>
<tr>
<td><code>-K REALOS</code></td>
<td>Effects in-line expansion for the ITRON system call function</td>
</tr>
<tr>
<td>`-K {SIZE</td>
<td>SPEED}`</td>
</tr>
<tr>
<td>`-K {UCHAR</td>
<td>SCHAR}`</td>
</tr>
<tr>
<td>`-K {UBIT</td>
<td>SBIT}`</td>
</tr>
<tr>
<td>`-kanji {SJIS</td>
<td>EUC}`</td>
</tr>
<tr>
<td><code>-O level</code></td>
<td>Gives instructions for general-purpose optimization</td>
</tr>
</tbody>
</table>
### 3.4 COMMAND OPTIONS

#### Table 3.4-1 List of Command Options (Continued)

<table>
<thead>
<tr>
<th>Specifying Format</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-o pathname</code></td>
<td>Outputs the result to the pathname</td>
</tr>
<tr>
<td><code>-P</code></td>
<td>Performs preprocessing only and outputs the result to .i</td>
</tr>
<tr>
<td><code>-S</code></td>
<td>Performs processes up to compiling and outputs the result to .asm</td>
</tr>
<tr>
<td><code>-s defname=newname [, attr [, address]]</code></td>
<td>Changes the section name</td>
</tr>
<tr>
<td><code>-T item, arg1 [, arg2 ...]</code></td>
<td>Passes arguments to the tool</td>
</tr>
<tr>
<td><code>-U name</code></td>
<td>Cancels the macro name definition</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Outputs the executed compiler tool version information to the standard output</td>
</tr>
<tr>
<td><code>-w level</code></td>
<td>Specifies the warning message output level</td>
</tr>
<tr>
<td><code>-Xdof</code></td>
<td>Inhibits the default option file read operation</td>
</tr>
<tr>
<td><code>-x func [, fun2 ...]</code></td>
<td>Specifies the in-line expansion of functions</td>
</tr>
<tr>
<td><code>-xauto [size]</code></td>
<td>Specifies the in-line expansion of the functions whose logical line count is not less than size</td>
</tr>
<tr>
<td><code>-Y item, dir</code></td>
<td>Changes the item position to dir</td>
</tr>
</tbody>
</table>

#### Table 3.4-2 List of Command Options

<table>
<thead>
<tr>
<th>Specifying Format</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-div905</code></td>
<td>Specifies the DIV/DIVW instruction is generated</td>
</tr>
<tr>
<td><code>-K ACCOPT</code></td>
<td>The optimization of accumulator transfer code for the immediate value.</td>
</tr>
<tr>
<td><code>-K ADDSP</code></td>
<td>Releases argument areas altogether</td>
</tr>
<tr>
<td><code>-K ARRAY</code></td>
<td>Optimization of array element access code</td>
</tr>
<tr>
<td><code>-K BITOPT</code></td>
<td>Effective generation of the bit operation instruction.</td>
</tr>
<tr>
<td><code>-K BITFIELD_ORDER_LSB</code></td>
<td>The bit-field member is arranged from the LSB side.</td>
</tr>
<tr>
<td><code>-K BITFIELD_ORDER_MSB</code></td>
<td>The bit-field member is arranged from the MSB side.</td>
</tr>
<tr>
<td><code>-pack</code></td>
<td>Packing of struct and union members.</td>
</tr>
<tr>
<td>`-model {SMALL</td>
<td>MEDIUM</td>
</tr>
<tr>
<td><code>-ramconst</code></td>
<td>Specifies that the mirror function will not be used</td>
</tr>
<tr>
<td>`-varorder {SORT</td>
<td>NORMAL}`</td>
</tr>
<tr>
<td>`-align {DIR1</td>
<td>DIR2}`</td>
</tr>
<tr>
<td><code>-rp</code></td>
<td>Specifies to change function call interface to register argument passing.</td>
</tr>
</tbody>
</table>
3.4.2 List of Command Cancel Options

The listed options are used to cancel command options on an individual basis. The cancel options for the command are listed in Table 3.4-3 and Table 3.4-4.

**List of Command Cancel Options**

<table>
<thead>
<tr>
<th>Specifying Format</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XB</td>
<td>Cancels the -B option</td>
</tr>
<tr>
<td>-XC</td>
<td>Cancels the -C option</td>
</tr>
<tr>
<td>-Xcmsg</td>
<td>Cancels the -cmsg option</td>
</tr>
<tr>
<td>-Xcwno</td>
<td>Cancels the -cwno option</td>
</tr>
<tr>
<td>-Xf</td>
<td>Cancels the -f option</td>
</tr>
<tr>
<td>-Xg</td>
<td>Cancels the -g option</td>
</tr>
<tr>
<td>-XH</td>
<td>Cancels the -H option</td>
</tr>
<tr>
<td>-Xhelp</td>
<td>Cancels the -help option</td>
</tr>
<tr>
<td>-XI</td>
<td>Cancels the -I option</td>
</tr>
<tr>
<td>-INF NOLINENO</td>
<td>Cancels the LINENO suboption</td>
</tr>
<tr>
<td>-INF NOLIST</td>
<td>Cancels the LIST suboption</td>
</tr>
<tr>
<td>-INF NOSRCIN</td>
<td>Cancels the SRCIN suboption</td>
</tr>
<tr>
<td>-INF NOSTACK</td>
<td>Cancels the STACK suboption</td>
</tr>
<tr>
<td>-K ALIAS</td>
<td>Cancels the NOALIAS suboption</td>
</tr>
<tr>
<td>-K INTLIB</td>
<td>Cancels the NOINTLIB suboption</td>
</tr>
<tr>
<td>-K NOEOP</td>
<td>Cancels the EOPT suboption</td>
</tr>
<tr>
<td>-K NOLIB</td>
<td>Cancels the LIB suboption</td>
</tr>
<tr>
<td>-K NOREALOS</td>
<td>Cancels the REALOS suboption</td>
</tr>
<tr>
<td>-K UNROLL</td>
<td>Cancels the NOUNROLL suboption</td>
</tr>
<tr>
<td>-K VOLATILE</td>
<td>Cancels the NOVOLATILE suboption</td>
</tr>
<tr>
<td>-Xo</td>
<td>Cancels the -o option</td>
</tr>
<tr>
<td>-Xs</td>
<td>Cancels the -s option</td>
</tr>
<tr>
<td>-XT item</td>
<td>Cancels the -T item specifying</td>
</tr>
<tr>
<td>-XV</td>
<td>Cancels the -V option</td>
</tr>
<tr>
<td>-Xx</td>
<td>Cancels the -x option</td>
</tr>
<tr>
<td>-Xxauto</td>
<td>Cancels the -xauto option</td>
</tr>
<tr>
<td>-XY item</td>
<td>Cancels the -Y item specifying</td>
</tr>
</tbody>
</table>
Table 3.4-4 List of Command Cancel Options

<table>
<thead>
<tr>
<th>Specifying Format</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-K NOACCOPT</td>
<td>Cancels the ACCOPT suboption</td>
</tr>
<tr>
<td>-K NOADDSP</td>
<td>Cancels the ADDSP suboption</td>
</tr>
<tr>
<td>-K NOARRAY</td>
<td>Cancels the ARRAY suboption</td>
</tr>
<tr>
<td>-K NOBITOPT</td>
<td>Cancels the BITOPT suboption</td>
</tr>
<tr>
<td>-Xpack</td>
<td>Cancels the -pack option</td>
</tr>
<tr>
<td>-Xdiv905</td>
<td>Cancels the -div905 option</td>
</tr>
<tr>
<td>-Xramconst</td>
<td>Cancels the -ramconst option</td>
</tr>
<tr>
<td>-Xalign</td>
<td>Cancels the -align option</td>
</tr>
<tr>
<td>-Xrp</td>
<td>Cancels the -rp option</td>
</tr>
</tbody>
</table>
CHAPTER 3 OPERATION

3.5 DETAILS OF OPTIONS

This section details the options.

■ Translation Control Related Options
  The translation control related options are related to preprocessor, compiler and assembler call
  control.

■ Preprocessor Related Options
  The preprocessor related options are related to preprocessor operations.

■ Data Output Related Options
  The data output related options are related to the command, preprocessor, and compiler data
  outputs.

■ Language Specification Related Options
  The language specification related options are related to the specification of the language to be
  recognized by the compiler.

■ Optimization Related Options
  The optimization related options are related to the optimization to be effected by the compiler.

■ Output Object Related Options
  The output object related options are related to the output object format.

■ Debug Information Related Options
  The debug information related options are related to the debug information to be referred by the
  symbolic debugger.

■ Command Related Options
  The command related options are related to the other tools recalled by commands.

■ Option File Related Options
  The option file related options are related to option files.
3.5 DETAILS OF OPTIONS

3.5.1 Translation Control Related Options

This section describes the options related to preprocessor, compiler and assembler call control.

Translation Control Related Options

The priorities of the translation control related options are defined as follows. They are not related to the order of specifying.

-E > -P > -S

The translation control related option exclusiveness is shown in Table 3.5-1.

Table 3.5-1 Translation Control Related Option Exclusiveness

<table>
<thead>
<tr>
<th>Specified Option</th>
<th>Option Invalidated</th>
</tr>
</thead>
<tbody>
<tr>
<td>-E</td>
<td>-S</td>
</tr>
<tr>
<td>-P</td>
<td>-S</td>
</tr>
<tr>
<td>-S</td>
<td>None</td>
</tr>
</tbody>
</table>

If the -E and -P options are specified simultaneously, see the explanation below.

-C option cannot use by the fcc907s command.

The translation control related options are detailed below.

- **-E**

   This option subjects all files to preprocessing only and outputs the result to the standard output. The output result contains the preprocessing instruction generated by the preprocessor, which is necessary for the compiler. The information targets for the preprocessing instruction generated by the preprocessor are the #line and #pragma instructions. If the -P option is specified together with the -E option, the preprocessing instruction generated by the preprocessor is inhibited. If the input file is not a C source file, the -E option does not do anything.

   [Example]
   
   ```
   >fcc907s -E -cpu MB90F553A sample.c
   
   The sample.c preprocessing result is output to the standard output.
   ```

- **-P**

   This option subjects a C source file to preprocessing only and outputs the result to the file whose extension is changed to .i. Unlike the cases where the -E option is specified, the output result does not contain the preprocessing instruction generated by the preprocessor. If the input file is not a C source file, the -P option does not do anything.

   [Example]
   
   ```
   >fcc907s -P -cpu MB90F553A sample.c
   
   The sample.c preprocessing result is output to the sample.i.
CHAPTER 3  OPERATION

- **-S**

This option performs processes up to compiling and outputs the resultant assembler source to file extension changed to .asm. If the input file is neither a C source file nor a preprocessed C source file, the -S option does not do anything.

[Example]

> fcc907s -S -cpu MB90F553A sample.c

The sample.c preprocessing and compiling process result are output to the sample.asm.

The relationship among file types, translation control related options, and processes is shown in Table 3.5-2.

**Table 3.5-2  Relationship Among File Types, Translation Control Related Options, and Processes**

<table>
<thead>
<tr>
<th>Option File Type (Extension)</th>
<th>-E</th>
<th>-P</th>
<th>-S</th>
<th>Nothing Specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source file (.c)</td>
<td>P</td>
<td>P</td>
<td>P C</td>
<td>P, C and A</td>
</tr>
<tr>
<td>Preprocessed C source file (.i)</td>
<td></td>
<td></td>
<td>C</td>
<td>C and A</td>
</tr>
<tr>
<td>Assembler source file (.asm)</td>
<td></td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Object file (.obj)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The fcc907s command does not call linker.

[Example]

> fcc907s -E file1.c file2.i -cpu MB90F553A

Subjects a file named file1.c to preprocessing only and outputs the result to the standard output. Performs nothing for a file named file2.i.

> fcc907s -S file1.c file2.i file3.asm -cpu MB90F553A

Subjects a file named file1.c to preprocessing and compiling and a file named file2.i to compiling. Performs nothing for a file named file3.asm. As a result, files named file1.asm and file2.asm are generated in the current directory.
3.5.2 Preprocessor Related Options

This section describes the options related to preprocessor operations. If the preprocessor is not called, the preprocessor related options are invalid.

- Preprocessor Related Options

The preprocessor related options are detailed below.

- **-B**

- **-XB**

  The -B option allows C++ style comments. When specifying this option, // style in addition to /* */ style can be used.

  The -XB option cancels the -B option.

  [Using Example]

  ```
  /* Comment */
  // Comment
  ```

- **-C**

- **-XC**

  The -C option retains all comments except those which are in the preprocessing instruction line as the preprocessing result. If the option is not specified, the comments are replaced by one blank character.

  The -XC option cancels the -C option.

  [Output Example]

  - Input:
    ```
    /* Comment */
    void func(void);
    ```
  - Operation:
    ```
    fcc907s -C -E -cpu MB90F553A sample.c
    ```
  - Output:
    ```
    # 1 "test5.c"
    /* Comment */
    void func(void);
    ```

- **-D name [=tokens]**

  This option defines the macro name with the tokens used as the macro definition. The option is equivalent to the following #define instruction.

  ```
  #define name  tokens
  ```

  If =tokens entry is omitted, the value 1 is given as the tokens value. If the tokens entry is omitted, the specified lexeme is deleted from the source file. The error related to the -D option is the same as the error related to the #define instruction. This option can be specified more than one time.
[Example]

```bash
>fcc907s -D os=m -D sys file.c -cpu MB90F553A
```

In a file named file.c, processing is conducted on the assumption that the macro definitions for os and sys are m and 1, respectively.

- `-H`
- `-XH`

The `-H` option outputs the header file pathnames acquired during preprocessing to the standard output. The pathnames are sequentially outputted, one for each line, in the order of acquisition. If there are any two exactly the same pathnames, only the first one will be outputted. When this option is specified, the command internally sets up the `-E` option to subjects all files to preprocessing only. However, the preprocessing result will not be outputted.

The `-XH` option cancels the `-H` option.

[Output Example]

- **Input:**
  ```
  #include <stdio.h>
  #include "head.h"
  ```

- **Operation:**
  ```bash
  fcc907s -H -cpu MB90F553A sample.c
  ```

- **Output:**
  ```
  /usr/softune/lib/907/include/stdio.h
  /usr/softune/lib/907/include/stddef.h
  /usr/softune/lib/907/include/stdarg.h
  ./head.h
  ```

- `-I dir`
- `-XI`

The `-I` option changes the rule of header file search so that the directory specified by `dir` will be searched prior to the standard directory. The standard directory is `$INC907$`.

This option can be specified more than one time. The search will be conducted in the order of specifying. When the option is specified, the header file search will be conducted in the following directories in the order shown below.

- Header file enclosed within angular brackets (`< >`)
  1. Directory specified by the `-I` option
  2. Standard directory

- Header file enclosed by double quotation marks (`" `)
  1. Directory having a file containing the `#include` line
  2. Directory specified by the `-I` option
  3. Standard directory

If a header file is specified by specifying its absolute path name, only the specified absolute path name will be searched. If any nonexistent directory is specified, this option is invalid.

The `-XI` option cancels the `-I` option.
-U name

This option cancels the macro name definition specified by -D. The option is equivalent to the following #undef instruction.

```c
#undef name
```

If the same name is specified by the -D and -U options, the name definition will be canceled without regard to the order of option specifying.

This option can be specified more than one time.

The error related to the -U option is the same as the error related to the #undef instruction.

[Example]

```bash
>fcc907s -U m -D n -D m file.c -cpu MB90F553A
```

This will cancel the macro m definition specified by the -D option.
CHAPTER 3 OPERATION

3.5.3 Data Output Related Options

This section describes the options related to the command, preprocessor, and compiler data outputs.

- Data Output Related Options
  - -cmsg
    This option outputs the compiling process completion message.
    [Example]
    - Operation:
      fcc907s -cmsg -S -cpu MB90F553A sample.c
    - Output:
      COMPLETED C Compile, FOUND NO ERROR : sample.c
  - -cwno
    This option sets the end code to 1 when a warning-level error occurs. When the option is not specified, the end code is 0.
  - -help
  - -Xhelp
    The -help option outputs the option list to the standard output. The -Xhelp option cancels the -help option.
    [Example]
    >fcc907s -help
    Various command option lists are outputted to the standard output.
  - -INF LINENO
  - -INF NOLINENO
    The -INF LINENO option inserts C source file line numbers into the assembler source file as comments. The LINENO suboption cannot be specified simultaneously with the SRCIN suboption.
    The NOLINENO suboption cancels the LINENO suboption.
3.5 DETAILS OF OPTIONS

[Output Example]

- Input:
  ```
  void func(void){}
  ```

- Operation:
  ```
  fcc907s -INF lineno -S -cpu MB90F553A sample.c
  ```

- Output:
  ```
  _func:
      LINK    #0
      ;;;;     sample.c, line 1
      UNLINK
      RET
  ```

- -INF LIST
- -INF NOLIST

The -INF LIST option generates a file in the current directory and outputs the assemble list. The name of the generated file is determined by changing the source file name extension to .lst. Since the assemble list is generated at assembling, it is not generated when assembling is not conducted. For the details of the assemble list, refer to the Assembler Manual.

The NOLIST suboption cancels the LIST suboption.

[Example]

```
>fcc907s -INF list -cpu MB90F553A sample.c
```

The sample.c preprocessing, compiling, and assembling process result are outputted to the sample.obj, and the resulting assemble list is outputted to the sample.lst.

- -INF SRCIN
- -INF NOSRCIN

The -INF SRCIN option inserts a C source file into the assembler source file as a comment. The NOSRCIN suboption cancels the SRCIN suboption.

The SRCIN suboption cannot be specified simultaneously with the LINENO suboption.

[Output Example]

- Input:
  ```
  void func(void){}
  ```

- Operation:
  ```
  fcc907s -INF srcin -S -cpu MB90F553A sample.c
  ```

- Output:
  ```
  _func:
      LINK    #0
      ;;;;     void func(void){}
      UNLINK
      RET
  ```
CHAPTER 3 OPERATION

- **INF STACK** [=file]

- **INF NOSTACK**

  The **INF STACK** [=file] option generates the specified file in the current directory and outputs the stack use amount data. If no file is specified, the information in all the simultaneously compiled files is output into files whose names are determined by changing the source file extensions to .stk.

  If the **-K ADDSP** option is simultaneously specified, stacks will not successively be freed so that the generated stack use amount data is inaccurate. In such an instance, therefore, it is well to remember that the maximum stack use amount data calculated by the SOFTUNE C ANALYZER may be smaller than the actual maximum use amount. For stack use amount data utilization procedures and data file specifications, refer to the SOFTUNE C ANALYZER MANUAL.

  The **NOSTACK** suboption cancels the **STACK** suboption.

  [Output Example]

  - **Input:**
    
    ```c
    extern void sub(void);
    void func(void){sub();}
    ```

  - **Operation:**
    
    ```bash
    fcc907s -INF stack -S -cpu MB90F553A sample.c
    ```

  - **Output:**
    
    ```bash
    @sample.c
    # E=Extern  S=Static  I=Interrupt
    # {Stack}  {E|S|I} {function name} [A]
    #  ->  {E_S}  {call function}
    # ...
    #
    #
    4   E   _func
    ->   E   _sub
    ```

- **-o pathname**

- **-Xo**

  The **-o pathname** option uses the pathname as the output file name. If this option is not specified, the default for the employed file format is complied with.

  The **-Xo** option cancels the **-o** option.

  [Example]

  ```bash
  >fcc907s -o output.asm -S -cpu MB90F553A sample.c
  ```

  The sample.c preprocessing and compiling process result are outputted to the output.asm.

- **-V**

- **-XV**

  The **-V** option outputs the version information about each executed compiler tool to the standard output. The **-XV** option cancels the **-V** option.
-w level

This option specifies the output level of warning-type diagnostic messages. Levels 0 through 8 can be specified. When level 0 is specified, no warning messages will be generated. The greater the level value, the more warning messages will be generated.

If the -w level option is not specified, -w 1 applies.

For the details of diagnostic messages, see 3.7, “MESSAGES GENERATED IN TRANSLATION PROCESS”.

For the relationship between warning level and warning item, see Table 3.5-3.

[Output Example]

• Input:
  
  const int a;

• Operation:
  
  fcc907s -w 5 -S -cpu MB90F553A sample.c

• Output:
  
  *** a.c(1) W1219C: ‘const’ a is not initialized.
### Table 3.5-3 Warning item at each warning level

<table>
<thead>
<tr>
<th>Warning level</th>
<th>Warning item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>Warning-type diagnostic message is not generated.</td>
</tr>
<tr>
<td>Level 1</td>
<td>A basic warning-type diagnostic messages are generated.</td>
</tr>
</tbody>
</table>
| Level 2       | The following warning-type diagnostic messages in addition to level 1 are generated.  
|               | Warning of the variable not used in the function is generated.             |
|               | Warning of the variable used before being initialized in the function is generated. |
|               | Warning of the presence of the use of the Static function is generated.     |
| Level 3       | The following warning-type diagnostic messages in addition to level 2 are generated.  
|               | When there is no return in the function which should return the value, warning is generated. |
|               | When the value is not specified for return by the function which should return the value, warning is generated. |
|               | Warning of pragma which cannot be recognized is generated.                 |
|               | When the variable and the constant are compared in the comparison operation, warning of the range for the constant value is generated. |
| Level 4       | The following warning-type diagnostic messages in addition to level 3 are generated.  
|               | When the extern function is declared in the block, warning is generated.    |
|               | When the struct/union is not defined in the extern declaration of the struct/union array, warning is generated. |
|               | When not the relational expression but the assignment expression, etc. are described in the place where the conditional expression is expected, warning is generated. |
|               | When the address of the auto variable is used as a return value of the function, warning is generated. |
| Level 5       | The following warning-type diagnostic messages in addition to level 4 are generated.  
|               | When there is an implicit int type declaration, warning is generated.      |
|               | When there is no prototype declaration of the function, warning is generated. |
|               | When the constant is described in the condition expression, warning is generated. |
|               | When there is an implicit int type declaration of the parameter, warning is generated. |
|               | When the declaration overload the declaration before, warning is generated. |
|               | When the comma continues at enum member’s end, warning is generated.       |
|               | When there is no initial value in the declaration with const, warning is generated. |
|               | When the address of the variable is compared with 0, warning is generated.   |
|               | When the type is defined in the cast expression, warning is generated.      |
|               | When register is specified for struct, union, and the array variable declaration, warning is generated. |
| Level 6       | The following warning-type diagnostic messages in addition to level 5 are generated.  
|               | When there is switch statement which is not default, warning is generated.  |
| Level 7       | The following warning-type diagnostic messages in addition to level 6 are generated.  
|               | When the int type is used, warning is generated.                           |
|               | When the bit field is neither int, signed int nor unsigned int type, warning is generated. |
| Level 8       | The following warning-type diagnostic messages in addition to level 7 are generated.  
|               | When the function is called with a pointer to the function, warning is generated. |
3.5 DETAILS OF OPTIONS

3.5.4 Language Specification Related Options

This section describes the options related to the specifications of the language to be recognized by the compiler.

- Language Specification Related Options
  - -J {a|c}
    This option specifies the language specification level to be interpreted by the compiler (preprocessor included).
    When -Ja is specified, interpretation is conducted in compliance with the ANSI standard including expansion specifications.
    When -Jc is specified, interpretation is conducted in strict compliance with the ANSI standard.
    In response to the expansion specifications, a warning message is outputted.
    If the option is not specified, -Ja applies.
    [Example]
    ```
    >fcc907s -J a file1.c -J c file2.c -cpu MB90F553A
    The -Jc option becomes valid so that files named file1.c and file2.c are interpreted in strict compliance with the ANSI standard.
    ```
  - -K {DCONST|FCONST}
    When the FCONST suboption is specified, a floating-point constant whose suffix is not specified will be handled as a float type.
    When the DCONST suboption is specified, a floating-point constant whose suffix is not specified will be handled as a double type.
    If neither of the above two suboptions is specified, -K DCONST applies.
    [Output Example]
    • Input:
      ```
      extern float f1,f2;
      void func(void)( f1 = f2+1.0;)
      ```
    • Operation:
      ```
      fcc907s -K fconst -cpu MB90F553A -S sample.c
      ```
The NOINTLIB suboption calls a normal function without effecting in-line expansion of an interrupt related function (__DI(), __EI(), and __set_il()). The INTLIB suboption cancels the NOINTLIB suboption.

[Output Example]

Input: 
void func(void) { __DI(); }

Operation: 
fcc907s -K nointlib -cpu MB90F553A -S sample.c

Output: 
_func:
    LINK    #0
    CALL    ___DI
    UNLINK
    RET

-The NOVOLATILE suboption does not recognize a __io qualifier attached variable as a volatile type. Therefore, __io qualifier attached variables will be optimized.

The VOLATILE suboption cancels the NOVOLATILE suboption.

[Example]
>fcc907s -K novolatile -S -O -cpu MB90F553A sample.c

When an __io qualifier attached variable is processed in sample.c, it is not handled as a volatile qualifier attached variable, but is treated as the optimization target.
3.5 DETAILS OF OPTIONS

- **-K \{UCHAR|SCHAR\}**
  
  This option specifies whether or not to treat the char type most significant bit as a sign bit. When the UCHAR suboption is specified, the most significant bit will not be treated as a sign bit. When the SCHAR suboption is specified, the most significant bit will be treated as a sign bit.
  
  If neither of the above two suboptions is specified, -K UCHAR applies.
  
  [Output Example]
  
  - Input:
    
    ```
    extern int data;
    char c = -1;
    void func(void){ data = c;}
    ```
    
  - Operation:
    
    ```
    fcc907s -K schar -cpu MB90F553A -S sample.c
    ```
    
  - Output:
    
    ```
    MOVX A, _c
    MOVW _data, A
    ```

- **-K REALOS**

- **-K NOREALOS**

  The REALOS suboption effects in-line expansion of the ITRON system call function. It can be used in cases where a program running under REALOS is to be prepared. For the ITRON system call function, refer to the REALOS/907 Kernel Manual.
  
  When specifying the REALOS suboption, be sure to include the system call declaration header file provided by the REALOS. If the REALOS suboption is specified without including the system call declaration header file and system call in-line expansion is initiated, the operation is not guaranteed, because it is possible that an adequate argument-type check has not been completed.
  
  The NOREALOS suboption cancels the REALOS suboption.
  
  [Output Example]
  
  - Input:
    
    ```
    #include "scdef_w.h"
    void func(void){ ext_tsk();}
    ```
    
  - Operation:
    
    ```
    fcc907s -K realos -cpu MB90F553A -S sample.c
    ```
    
  - Output:
    
    ```
    INTP ext_tsk
    BRA *
    ```

- **-K \{UBIT|SBIT\}**

  This option specifies whether or not to treat the most significant bit as a sign bit in situations where the char, short int, int, or long int type is selected as the bit field. When the UBIT suboption is specified, the most significant bit will not be treated as a sign bit. When the SBIT suboption is specified, the most significant bit will be treated as a sign bit.
  
  If neither of the above two suboptions is specified, -K UBIT applies.
[Output Example]

• Input:
  extern int data;
  struct tag { int bf:1;}st = {-1};
  void func(void)( data = st.bf;)

• Operation:
  fcc907s -K sbit -cpu MB90F553A -S sample.c

• Output:
  MOVB A, _st:0
  EXT
  MOVW _data, A
3.5.5 Optimization Related Options

This section describes the options related to optimization by the compiler.

- Optimization Related Options
  - -K SIZE
    This option selects an appropriate optimization combination with emphasis placed upon the object size. The available options are shown below.
    -O 3
    -K EOPT
    -K NOUNROLL
    If any option (e.g., -O0) contradictory to the these options is specified after the SIZE suboption, such a contradictory option takes effect.
    The -K SIZE option not only offers the optimization combination selection function, but also makes it possible to issue a generation instruction for object size minimization and effect object pattern switching.
  - -K SPEED
    This option selects an appropriate optimization combination with emphasis placed upon the generated object execution speed. The available options are shown below.
    -O 4
    If any option (e.g., -O0) contradictory to the these options is specified after the SPEED suboption, such a contradictory option takes effect.
    The -K SPEED option not only offers the optimization combination selection function, but also makes it possible to issue a generation instruction for execution speed maximization and effect object pattern switching.
  - -O [level]
    This option specifies the optimization level. Levels 0, 1, 2, 3, and 4 can be specified. The higher the optimization level performs, the shorter the generated object execution time becomes, but the longer the compilation time takes. Note that higher optimization level contains lower optimization level functions.
    One of the following levels is to be specified. When no level is specified, -02 applies.
    -O: Optimization Level 0
      No optimization will be effected. This level is equivalent to cases where the -0 is not specified.
    -1: Optimization Level 1
      Optimization will be effected in accordance with detailed analyses of a program flow.
    -2: Optimization Level 2
      The following optimization feature is exercised in addition to the feature provided by optimization level 1.
CHAPTER 3 OPERATION

- Loop Unrolling

Loop unrolling is performed to increase the execution speed by decreasing the loop count when loop-count detection is possible. However, it tends to increase object size. Therefore, this optimization should not be used in situations where object size is important.

[Before Unrolling]
for(i=0;i<3;i++){ a[i]=0;}

[After Unrolling]
a[0]=0;
a[1]=0;
a[2]=0;

-3: Optimization Level 3

The following optimization features are exercised in addition to the features provided by optimization level 2.

- Loop Unrolling (Extended)

Loops, including branch instructions, that have not been the target of optimization level-2 loop unrolling, are the target of this extended loop unrolling.

- Optimization Function Repeated Execution

In optimization function repeated execution, the optimization features except the loop unrolling feature will be repeatedly executed until no more optimization is needed. However, the translation time will increase.

-4: Optimization Level 4

The following optimization features are exercised in addition to the features provided by optimization level 3.

- Arithmetic Operation Evaluation Type Change (same as effected by -K EOPT specifying)

Performs optimization to change arithmetic operation evaluation type at compilation stage. When this option is specified, there may be side effects on the execution results.

- Standard Function Expansion/Change (same as effected by -K LIB specifying)

Switches to a higher-speed standard function that recognizes standard function operations, performs standard function in-line expansion, and performs identical operations. When this option is specified, there may be side effects on the execution results. Since standard function in-line expansion is implemented, the code size may increase.

○ -K ADDSP

○ -K NOADDSP

The -K ADDSP option releases argument areas placed in the stacks for function calling. Since the argument areas are released altogether for optimization purposes, the function calling overhead decreases so that a smaller, higher-speed object results.

When -K ADDSP is specified, the stacks will not successively be released. Therefore, the stack use amount data, which is generated upon -INF STACK option specifying, will be inaccurate. In such an instance, it is well to remember that the maximum stack use amount data calculated by the SOFTUNE C ANALYZER may be smaller than the actual maximum use amount.

The NOADDSP suboption cancels the ADDSP suboption.
3.5 DETAILS OF OPTIONS

[Output Example]

• Input:

```c
extern int i;
extern void sub(int);
void func(void){
  sub(i);
  sub(i);
}
```

• Operation:

```bash
fcc907s -K addsp -cpu MB90F553A -S sample.c
```

• Output:

```asm
MOVW    A, _i
PUSHW   A
CALL    _sub
MOVW    A, _i
PUSHW   A
CALL    _sub
ADDSP   #4; Releasing argument areas synthesized
```

- K EOPT

- K NOEOPT

The EOPT suboption effects optimization by changing the arithmetic operation evaluation type at the compilation stage. When this option is specified, there may be side effects on the execution results. This option takes effect only when it is specified simultaneously with the -O option.

The NOEOPT suboption cancels the EOPT suboption.

[Output Example]

• Input:

```c
extern int i;
void func(int a, int b){
  i=a-100+b+100;
}
```

• Operation:

```bash
fcc907s -K eopt -O -cpu MB90F553A -S sample.c
```

• Output:

```asm
MOVW    A, @RW3+4
ADDW    A, @RW3+6 ; Order of arithmetic operation replaced
MOVW    _i, A
```
CHAPTER 3 OPERATION

❖ -K LIB

❖ -K NOLIB

The LIB suboption recognizes the standard function operation and replaces the standard function with a higher-speed standard function which effects standard function in-line expansion and performs the same operation as the original standard function. When this option is specified, there may be side effects on the execution results. Since standard function in-line expansion is implemented, the code size may increase. This option takes effect only when it is specified simultaneously with the -O option.

The NOLIB suboption cancels the LIB suboption.

[Output Example]

• Input:
  extern int i;
  void func(void){
    i=strlen("ABC");
  }

• Operation:
  fcc907s -K lib -O -cpu MB90F553A -S sample.c

• Output:
  MOVN A, #3 ; Processing equivalent to strlen expanded
  MOVW _i, A

❖ -K NOALIAS

❖ -K ALIAS

The NOALIAS suboption optimizes the data specified by the pointer on the assumption that the pointer does not specify the same area as the other variables or pointers. This option takes effect only when it is specified simultaneously with the -O option. The language specification permits the pointer to point to the same area as any other variable or pointer. Therefore, when using this option, check the program carefully.

The ALIAS suboption cancels the NOALIAS suboption.

[Output Example]

• Input:
  extern int i;
  extern int j;
  void func9(int *p){
    *p=i+1;
    j=i+1;
  }

• Operation:
  fcc907s -K noalias -O -cpu MB90F553A -S sample.c
• Output:

    MOVW A, _i 
    MOVN A, #1 
    ADDW A 
    MOVW RW4, A 
    MOVW A, @RW3+4 
    MOVW @AL, AH 
    MOVW A, RW4 
    MOVW _j, A ; Value of *p=i+1 reused

❍ -K NOUNROLL

❍ -K UNROLL

The NOUNROLL suboption inhibits loop unrolling optimization. Use this option when loop unrolling optimization is to be inhibited with the -O2 to -O4 options specified.

The UNROLL suboption cancels the NOUNROLL suboption.

❍ -x function name 1 [, function name 2, ...]

❍ -Xx

The -x option effects in-line expansion, instead of function calling, of functions defined by a C source. However, recursively called functions will not be subjected to in-line expansion. It should also be noted that functions may not be subjected to in-line expansion depending on asm statement use, structure/union type argument presence or absence, setjmp function calling, and other conditions. The option takes effect only when it is specified simultaneously with the -O option.

The -Xx option cancels the -x option.

[Output Example]

• Input:

    extern int a;
    static void sub(void){ a=1; }
    void func(void){ sub(); }

• Operation:

    fcc907s -cpu MB90F553A -O -x sub -S sample.c

• Output:

    _func:
    MOVN A, #1 
    MOVW _a, A 
    RET
CHAPTER 3 OPERATION

- **-xauto [size]**
  
  The -xauto option effects in-line expansion, instead of function calling, of functions whose logical line count is not less than size. However, recursively called functions will not be subjected to in-line expansion. It should also be noted that functions may not be subjected to in-line expansion depending on asm statement use, structure/union type argument presence or absence, setjmp function calling, and other conditions.

  If the size entry is omitted, the value 30 is assumed to be specified. The option takes effect only when it is specified simultaneously with the -O option.

  The -xauto option cancels the -xauto option.

- **-K ARRAY**
  
  The ARRAY suboption optimizes the array element access code (e.g. a[i]++;). This option takes effect only when it is specified simultaneously with the -O option. However, a part of optimization (e.g. deletion of dead variable) might be not effective when the option is specified and the code worsen according to the source program.

  The NOARRAY suboption cancels the ARRAY suboption.

- **-K ACCOPT**
  
  The ACCOPT suboption, continuous substitution of the same constant, is optimization of accumulator transfer code for the immediate value. The option takes effect only when it is specified simultaneously with the -O option.

  The NOACCOPT suboption cancels the ACCOPT suboption.

[Output Example]

- Input:
  
  ```c
  extern int a,b,c;
  void func(void){a=b=c=0;}
  ```

- Operation:
  
  ```bash
  fcc907s -K accopt -S -cpu MB90F553A -O sample.c
  ```

- Output:
  
  ```assembly
  _func:
  MOVN   A, #1
  MOVW   _c, A
  MOVW   _b, A
  MOVW   _a, A
  ```
3.5 DETAILS OF OPTIONS

- **-K BITOPT**

- **-K NOBITOPT**
  
  The **BITOPT** suboption does effective generation of the bit operation instruction. The option takes effect only when it is specified simultaneously with the **-O** option.

  The **NOBITOPT** suboption cancels the **BITOPT** suboption.

[Output Example]

- **Input:**
  
  ```c
  extern int a;
  void func(void){a|=0x80;}
  ```

- **Operation:**

  ```bash
  fcc907s -K bitopt -S -cpu MB90F553A -O sample.c
  ```

- **Output:**

  ```asm
  _func:
      SETB   _a:7
  ```
This section describes the options related to output object formats.

- **Output Object Related Options**

  - **-cpu MB number**

    In this option, the MB number of the CPU actually used is specified in the CPU information file. If the MB number not described in the CPU information file is specified, the compiler becomes an error because series information on the CPU is taken from the CPU information file.

    This option cannot be omitted.

    **Example**

    ```
    >fcc907s -S -cpu MB90F553A sample.c
    ```

  - **-div905**

  - **-Xdiv905**

    The -div905 option and the -Xdiv905 option are the options concerning the CPU bug of "DIV A,Ri" and "DIVW A,RWi" instructions of MB90500 series. This CPU bug is described to Appendix C "NOTES OF SIGNED DIVISION INSTRUCTION OF F2MC-16LX CPU".

    The -div905 option and the -Xdiv905 option can be specified only for the fcc907s command. And, only when the MB number of MB90500 series is specified by the -cpu option, these become effective.

    The -div905 option generates signed division instruction (DIV and DIVW). Please specify this option only when there is no problem even if the signed division instruction (DIV and DIVW) is used.

    The -Xdiv905 option cancels the -div905 option.

    When the -div905 option and the -Xdiv905 option are omitted to the specification for the MB number of MB90500 series for the -cpu option, the -Xdiv905 option is applied.

    When the -Xdiv905 option is specified, not the signed division instruction (DIV and DIVW) but Library Calls generated. Therefore, the amount of the stack use increases occasionally. Moreover, __mul(), __div(), and __mod() which is a built-in function are generated as not machine instructions but Library Calls.

  - **-model {SMALL|MEDIUM|COMPACT|LARGE}**

    This option specifies memory model. For the details of memory models, see 4.2, "MEMORY MODELS".

  - **-ramconst**

  - **-Xramconst**

    Specify this option (-ramconst) when the mirror function is not to be used. When specified, the option will position const-qualified static variables in the RAM.

    When this option is specified, the compiler generates the CINIT section corresponding to the CONST section, so that ROM data can be accessed with 16-bit symbols. The startup routine must copy the CONST internal data to the CINIT.
This option does not work on CONST_module name sections that are generated relative to large models, compact models, or __far-qualified variables.

The -Xramconst option cancels the -ramconst option.

[Output Example]

- Input:
  ```
  const int a=0x10;
  ```

- Operation:
  ```
  fcc907s -ramconst -S -cpu MB90F553A sample.c
  ```

- Output:
  ```
  .SECTION      CONST, CONST, ALIGN=2
  .ALIGN  2
  .DATA.H 16
  .SECTION      CINIT, DATA, ALIGN=2
  .ALIGN  2
  .GLOBAL _a

  _a:
  .RES.H 1
  ```

- `s defname=newname [, attr [, address]]`

- `s` defname=newname [, attr [, address]]

The -s option changes the compiler output section name from defname to newname, and changes section type to attr.

Large models, compact models, medium models, and __far-qualified variable or function section names can be specified by attaching FAR_ to the start.

The arrangement address can also be specified in the address position.

For compiler output section names, see 4.1, “SECTION STRUCTURE”. For selectable section types, refer to the Assembler Manual.

If the arrangement address is specified, the arrangement address cannot be specified relative to the associated section at linking.

The -Xs option cancels the -s option.

<Caution>

The operation is not guaranteed when the section having the location address is specified and the section having the location address is not specified exist together the same section name.

[Output Example]

- Input:
  ```
  void func(void){}
  ```

- Operation:
  ```
  fcc907s -s CODE=PROGRAM, CODE, 0x1000 -S -cpu MB90F553A sample.c
  ```

- Output:
-varorder (SORT|NORMAL)

This option specifies how external variables and static variables in a section are aligned. When SORT suboption is specified, to except the gap, external variables and static variables are aligned by the size of alignment. This has an effect to reduce the memory use amount. When NORMAL suboption is specified, external variables and static variables are aligned by the order of description. Variables specified __io qualifier are always aligned to the IO section by the order of description.

If neither of the above two suboptions is specified, -varorder SORT applies.

[Output Example]

- Input:
  - int i1;
  - char c;
  - int i2;

- Operation:
  - fcc907s -varorder NORMAL -S -cpu MB90F553A sample.c

- Output:
  - .SECTION DATA, DATA, ALIGN=2
  - .ALIGN 2
  - _i1: .RES.B 2
  - _c: .RES.B 1
  - .ALIGN 2
  - _i2: .RES.B 2

- pack

- Xpack

The -pack option packs the struct and union members.

The -Xpack option cancels the -pack option.

[Output Example]

- Input:
  - struct tag {
    - char a;
    - int b;
    - char c;
3.5 DETAILS OF OPTIONS

{ small;
  f() {s.b=0;}

- Operation:
  fcc907s -cpu MB90F553A -S -pack sample.c

- Output:
  MOVN  A, #0
  MOVW  _s+1, A

- -K BITFIELD_ORDER_LSB

- -K BITFIELD_ORDER_MSB

  The BITFIELD_ORDER_LSB suboption arranges bit-field member’s direction of arrangement from the LSB side. The BITFIELD_ORDER_MSB suboption arranges bit-field member’s direction of arrangement from the MSB side. When the BITFIELD_ORDER_MSB suboption is specified, __BITFIELD_ORDER_MSB__ is defined as a predefined macro. Please do not link objects which specify the BITFIELD_ORDER_MSB suboption with an object not so because bit-field member’s direction of arrangement is different.

- -align {DIR1|DIR2}

- -Xalign

  The -align option changes the boundary alignment value of __direct qualified variables. When the DIR1 suboption is specified, the boundary alignment value of __direct qualified variables is 1. When the DIR2 suboption is specified, the boundary alignment value of __direct qualified variables is 2.

  If neither of the above two suboptions is specified, the -align DIR2 is applied.

  The -Xalign option cancels the -align option.

[Output Example]

- Input:
  __direct int a;
  __direct char b;

- Operation:
  fcc907s -align DIR1 -cpu MB90F553A, -S sample.c

- Output:
  .SECTION  DIRDATA, DIR, ALIGN=1
  .GLOBAL  _a
  _a:
    .RES.B  2
    .GLOBAL  _b
  _b:
    .RES.B  1
CHAPTER 3 OPERATION

- -rp

- -Xrp

The -rp option changes the function call interface to “register argument passing” (the interface which passes the argument of function in the register (RW0, RW1)).

The -Xrp option cancels the -rp option.

If changing the HL function call interface, refer to “Appendix D GUIDE TO CHANGING FUNCTION-CALL INTERFACE”.

[Output Example]

- Input:
  ```
  int func(int a, int b){
    return a+b;
  }
  ```

- Operation:
  ```
  fcc907s -rp -O -cpu MB90F553A -S sample.c
  ```

- Output:
  ```
  _func:
    MOVW A, RW0 ;pass the argument a in the register
    ADDW A, RW1 ;pass the argument b in the register
    RET
  ```

<Caution>

The -rp option is specified, link the library with _rp. If the different objects of the function call interface are linked together, this operation is not guaranteed.
3.5.7 Debug Information Related Options

This section describes the options related to the debug information to be referred by the symbolic debugger.

■ Debug Information Related Options

- **-g**
- **-Xg**

The `-g` option adds debug data to the object file. To assure debugging accuracy, you should refrain from specifying the optimization option `-O[1-4]`. If the optimization option is specified, the compiler tries to assure better code output by changing the arithmetic operation target position and omitting any arithmetic operations that are judged to be unnecessary. To minimize the amount of data exchange with memory, the compiler tries to retain data within a register. It is therefore conceivable that a break point positioned in a certain line may fail to cause a break or that currently monitored certain address data may fail to vary with the expected timing. It also well to remember that the debug data will not be generated for an unused local variable or a local variable whose area need not be positioned in a stack as a result of optimization. Debugging must be conducted with the above considerations taken into account.

The `-Xg` option cancels the `-g` option.
CHAPTER 3 OPERATION

3.5.8 Command Related Options

This section describes the options related to the other tools called by the command.

- Command Related Options

- -Y item, dir
  - -XY
    - The -Y option changes the item position to the dir directory. The -XY option cancels the -Y option. The item is one of the following.
      - p: Changes the preprocessor pathname to dir
      - c: Changes the compiler pathname to dir
      - a: Changes the assembler pathname to dir
    - Example
      
      >fcc907s file.c -Yp,\home\newlib -cpu MB90F553A
      Calls the preprocessor using \home\newlib\cpps as the path name.

- -T item, arg1 [, arg2 ...]
  - -XT
    - The -T option passes arg to item as an individual compiler tool argument. The -XT option cancels the -T option.
    - Use a comma to separate arguments. To describe a comma as an argument, position a backslash (\) immediately before the comma. The comma positioned after the backslash will not be interpreted as a delimiter. To write a blank as an argument, describe a comma in place of a blank.
    - For the options for various commands, refer to the associated manuals. The following can be specified as the item.
      - a: Assembler
    - Example
      
      >fcc907s -Ta,-lf,asmlist file.c -cpu MB90F553A
      Sequentially passes arguments -lf and asmlist to the assembler. Therefore, the assemble list asmlist will be generated as a result of command execution.
3.5.9 Option File Related Options

This section describes the option file related options.

- Option File Related Options
  -f filename
  -Xf
  The -f option is used to read the specified option file (filename). If the option file name does not have an extension, an .opt extension will be added. The command options can be written in an option file. For the details of an option file, see 3.6, “OPTION FILES”.
  The -Xf option cancels the -f option.

-Xdof
This option specifies that the default option file will not be read. For the default option file, see 3.6, “OPTION FILES”.
This section describes command option files. With the option file feature, it is possible to specify a bunch of options written in a file. This feature also permits you to put startup options to be specified in a file.

■ Option File

Option file reading takes place when an associated option is specified. This assures that the same result is obtained as when an option is specified at the -f specifying position in the command line.

If the option file name is without an extension, an .opt extension will be added.

■ Option File General Format

All entries that can be made in a command line can be written in an option file.

A line feed in an option file is replaced by a blank.

A comment in an option file is replaced by a blank.

[Example]

-1 /usr/include # Include specifying
-D F^2MC16 # Macro specifying
-g # Debug data generation specifying
-S # Execution of processes up to compiling

■ Option File Limitations

The length of a line that can be written in an option file is limited to 4095 characters.

The -f option can be written in an option file. However, nesting is limited to 8 levels.

The Kanji character code in the option file should be the same as using the host's Kanji character code. The operation is not guaranteed when the Kanji character code on the command line and the Kanji character code in the option file are different.

<table>
<thead>
<tr>
<th>OS</th>
<th>Kanji character code which can be used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>ShiftJIS</td>
</tr>
<tr>
<td>Solaris2.x</td>
<td>EUC</td>
</tr>
<tr>
<td>HP-UX10.x</td>
<td>ShiftJIS</td>
</tr>
</tbody>
</table>
### Acceptable Comment Entry in Option File

A comment can be started from any column. A comment is to begin with a sharp (#). The entire remaining portion of the line serves as the comment. In addition, the following comments can also be used.

- /* Comment */
- // Comment
- ; Comment

[Example]

```
-l /usr/include  # Include specifying
-D F^2MC16 /* Macro specifying */
-g               // Debug data generation specifying
-S ; Execution of processes up to compiling
```

### Default Option File

A preselected option file can be read to initiate command execution. The obtained result will be the same as when an option is specified prior to another option specified in the command line. The default option file name is predetermined as follows.

[For UNIX OS]

```
${OPT907}/fcc907.opt
```

[For Windows]

```
>OPT907>\fcc907.opt
```

The default option file name is "fcc907.opt". If the default option file does not exist in the specified directory, such a specifying is ignored. To inhibit the default option file feature, specify the -Xdof option in the command line.
3.7 MESSAGES GENERATED IN TRANSLATION PROCESS

When an error is found in a source program or a condition which does not constitute a substantial error but requires attention is encountered, diagnostic messages may be generated at the time of translation. For message outputs generated by tools other than the compiler, refer to the respective manuals for the tool.

■ Messages Generated in Translation Process

A diagnostic message output example is shown in Figure 3.7-1.

*** test.c(4) E4110C: Identifier “a” is not declared.

Figure 3.7-1 Diagnostic Message Example

■ Tool Identifier

The tool identifier indicates the tool that has detected the error.

D: Command
P: Preprocessor
C: Compiler
A: Assembler

■ Error Level

The error level represents the diagnostic check result type.

Table 3.7-1 shows the relationship among various error levels, return codes, and their meanings.
### Table 3.7-1 Relationship between Error Levels and Return Codes

<table>
<thead>
<tr>
<th>Error Level</th>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>Indicates a condition which does not constitute an error but requires attention</td>
</tr>
</tbody>
</table>
| W           | 0           | Indicates a minor error  
Process execution continues without being interrupted. The return code can be changed by the -cwno option. |
| E           | 2           | Indicates a serious error  
Process execution stops. |
| F           | 3           | Indicates a fatal error which is related to quantitative limitations or system failure  
Process execution stops. |
CHAPTER 4  OBJECT PROGRAM STRUCTURE

This chapter describes the information necessary for program execution.

4.1  SECTION STRUCTURE
4.2  MEMORY MODELS
4.3  GENERATION RULES FOR NAMES USED BY COMPILER
4.4  BOUNDARY ALIGNMENT
4.5  BIT FIELD
4.6  STRUCTURE/UNION
4.7  FUNCTION CALL INTERFACE (STACK ARGUMENT PASSING)
4.8  FUNCTION CALL INTERFACE (REGISTER ARGUMENT PASSING)
4.9  INTERRUPT FUNCTION CALL INTERFACE
CHAPTER 4 OBJECT PROGRAM STRUCTURE

4.1 SECTION STRUCTURE

Table 4.1-1 shows the sections to be generated by the compiler and their meanings. When a section name is accessed using a 24-bit symbol, its name used is the section name plus the "_module name" attached to the end of the section name. (The section name specified by -s option becomes "FAR_SectionName".) The source file name is used as the module name. If the section name is changed by the -s option, the changed section name is used.

Section Structure

Table 4.1-1 Section List

<table>
<thead>
<tr>
<th>No.</th>
<th>Section Type</th>
<th>Section Name</th>
<th>Type</th>
<th>Boundary Alignment [Byte]</th>
<th>Write</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Code section</td>
<td>CODE</td>
<td>CODE</td>
<td>1</td>
<td>Disabled</td>
<td>Provided</td>
</tr>
<tr>
<td>2</td>
<td>Initialized section</td>
<td>INIT</td>
<td>DATA</td>
<td>2</td>
<td>Enabled</td>
<td>Not provided</td>
</tr>
<tr>
<td>3</td>
<td>Initial value of INIT</td>
<td>DCONST</td>
<td>CONST</td>
<td>2</td>
<td>Disabled</td>
<td>Provided</td>
</tr>
<tr>
<td>4</td>
<td>Constant section</td>
<td>CONST</td>
<td>CONST</td>
<td>2</td>
<td>Disabled</td>
<td>Provided</td>
</tr>
<tr>
<td>5</td>
<td>RAM area of CONST</td>
<td>CINIT</td>
<td>DATA</td>
<td>2</td>
<td>Enabled</td>
<td>Not provided</td>
</tr>
<tr>
<td>6</td>
<td>Data section</td>
<td>DATA</td>
<td>DATA</td>
<td>2 or 1</td>
<td>Enabled</td>
<td>Not provided</td>
</tr>
<tr>
<td>7</td>
<td>Initialized direct section</td>
<td>DIRINIT</td>
<td>DIR</td>
<td>2 or 1</td>
<td>Enabled</td>
<td>Not provided</td>
</tr>
<tr>
<td>8</td>
<td>Initial value of DIRINIT</td>
<td>DIRCONST</td>
<td>DIRCONST</td>
<td>2 or 1</td>
<td>Disabled</td>
<td>Provided</td>
</tr>
<tr>
<td>9</td>
<td>Direct section</td>
<td>DIRDATA</td>
<td>DIR</td>
<td>2</td>
<td>Enabled</td>
<td>Not provided</td>
</tr>
<tr>
<td>10</td>
<td>I/O section</td>
<td>IO</td>
<td>IO</td>
<td>2</td>
<td>Enabled</td>
<td>Not provided</td>
</tr>
<tr>
<td>11</td>
<td>Vector section</td>
<td>INTVECT</td>
<td>CONST</td>
<td>2</td>
<td>Disabled</td>
<td>Provided</td>
</tr>
<tr>
<td>12</td>
<td>Data transfer section</td>
<td>DTRANS</td>
<td>CONST</td>
<td>2</td>
<td>Disabled</td>
<td>Provided</td>
</tr>
<tr>
<td>13</td>
<td>Data clear section</td>
<td>DCLEAR</td>
<td>CONST</td>
<td>2</td>
<td>Disabled</td>
<td>Provided</td>
</tr>
</tbody>
</table>

The purpose of each section use and the relationship to the C language are explained below.

(1) Code section

Stores machine codes. This section corresponds to the procedure section for the C language.

The default section name is CODE.

(2) Initialized section

Stores the initial value attached variable area. For the C language, this section corresponds to the area for external variables without the const attribute, static external variables, and static internal variables.

The default section name is INIT.
(3) Initial value of INIT
Stores the initial values for initial value attached variables. This section is located in the
ROM. It is necessary to copy the DCONST data to the INIT using the startup routine. If the
order of section output by the compiler is changed to the detriment of DCONST-to-INIT
correspondence, no subsequent operations will be guaranteed.
The default section name is DCONST.

(4) Constant section
Stores the write-protected initial value attached variable area. For the C language, this
section corresponds to the area for const qualifier attached external variables, static external
variables, and static internal variables.
The default section name is CONST.

(5) RAM area of CONST
When the employed CPU type does not permit the use of the mirror function, this section can
be generated by specifying the -ramconst option. It is necessary to copy the CONST data to
the CINIT using the startup routine. If the order of section output by the compiler is changed
to the detriment of CONST-to-CINIT correspondence, no subsequent operations will be
guaranteed.
The default section name is CINIT.

(6) Data section
Stores the area for variables without the initial value. For the C language, this section
corresponds to the area for external variables (including those which are with the const
qualifier), static external variables, and static internal variables.
The default section name is DATA.

(7) Initialized direct section
Stores the area for __direct-qualified initial value attached variables. For the C language,
this section corresponds to the area for external variables, static external variables, and
static internal variables that are __direct-qualified and without the const qualifier.
The default section name is DIRINIT.

(8) Initial value of DIRINIT
Stores the initial values for the __direct-qualified initial value attached variables. This section
is located in the ROM. It is necessary to copy the DIRCONST data to the DIRINIT using the
startup routine. If the order of section output by the compiler is changed to the detriment of
DIRCONST-to-DIRINIT correspondence, no subsequent operations will be guaranteed.
The default section name is DIRCONST.

(9) Direct section
Stores the area for the __direct-qualified variables without the initial value. For the C
language, this section corresponds to the area for __direct-qualified external variables
(including those which are provided with the const qualifier), static external variables, and
static internal variables.
The default section name is DIRDATA.
(10) I/O section

Stores the area for the \_\_io-qualified variables. For the C language, this section corresponds to the area for \_\_io-qualified external variables (including those which are provided with the const qualifier), static external variables, and static internal variables. The default section name is IO.

(11) Vector section

Stores interrupt vector tables. For the C language, this section is generated only when the generation of a vector table is specified by \#pragma intvec.

The default section name is INTVECT.

(12) Data transfer/Data clear section

This section is data table for the initialization of the external variable area. For the details, see CHAPTER 6, "EXECUTION ENVIRONMENT".
4.2 MEMORY MODELS

This section describes the memory models.

- Memory Models

Table 4.2-1 shows the memory models selectable for compilation and their meanings. The compiler treats the code address and data address default set as a preselected memory model. In cases where a __far/ __near type qualifier is attached to a variable or function, the type qualifier specifying is complied with.

Table 4.2-1 List of Memory Models

<table>
<thead>
<tr>
<th>Memory Model</th>
<th>Code Address Space</th>
<th>Data Address Space</th>
<th>Compile Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small model</td>
<td>16 bit</td>
<td>16 bit</td>
<td>-model small</td>
</tr>
<tr>
<td>Medium model</td>
<td>24 bit</td>
<td>16 bit</td>
<td>-model medium</td>
</tr>
<tr>
<td>Compact model</td>
<td>16 bit</td>
<td>24 bit</td>
<td>-model compact</td>
</tr>
<tr>
<td>Large model</td>
<td>24 bit</td>
<td>24 bit</td>
<td>-model large</td>
</tr>
</tbody>
</table>

- Small Model

The small model is to be specified in situations where all codes and datas can be positioned within a 16-bit address space. Since all addresses are expressed using 16 bits, a compact, high-speed program can be realized.

When using a product without the mirror function, it is necessary to specify the -ramconst option for the purpose of securing a ROM data accessing area in RAM.

If the address size is specified by a type qualifier, such a specified address size is complied with.

When calling a __near type qualified function from a __far type qualified function, both functions must be positioned in the same section. The reason is that the PCB set up for __far type qualified function calling is used as is for __near type qualified function calling.

- Medium Model

The medium model is to be specified in situations where codes can be positioned in a 24-bit address space, and data can be positioned in a 16-bit address space.

When using a product without the mirror function, it is necessary to specify the -ramconst option for the purpose of securing a ROM data accessing area in RAM.

If the address size is specified by a type qualifier, such a specified address size is complied with.
CHAPTER 4 OBJECT PROGRAM STRUCTURE

Compact Model
The compact model is to be specified in situations where codes can be positioned in a 16-bit address space, and data can be positioned in a 24-bit address space.

If the address size is specified by a type qualifier, such a specified address size is complied with.

Variables have to be adjusted to the bank boundary. If not, the generated code cannot access such variable correctly.

Large Model
The large model is to be specified in situations where all codes and datas can be positioned in a 24-bit address space. Since all addresses are expressed using 24 bits, the codes used are redundant as compared to those for the small model.

If the address size is specified by a type qualifier, such a specified address size is complied with.

Variables have to be adjusted to the bank boundary. If not, the generated code cannot access such variable correctly.
This section describes the rules for the names used by the compiler.

### Generation Rules for Names Used by Compiler

Table 4.3-1 shows the relationship between the names generated by the compiler and the C language.

<table>
<thead>
<tr>
<th>C Language Counterpart</th>
<th>Label Generated by Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function name</td>
<td>_function name</td>
</tr>
<tr>
<td>External variable name</td>
<td>_external variable name</td>
</tr>
<tr>
<td>Static variable name</td>
<td>LI_no</td>
</tr>
<tr>
<td>Local variable name</td>
<td>—</td>
</tr>
<tr>
<td>parameter name</td>
<td>—</td>
</tr>
<tr>
<td>Character string, derived type</td>
<td>LS_no</td>
</tr>
<tr>
<td>Automatic variable initial value</td>
<td>LS_no</td>
</tr>
<tr>
<td>Target location label</td>
<td>L_no</td>
</tr>
</tbody>
</table>

**Note:** The compiler internal generation number is placed at the no position.
4.4 BOUNDARY ALIGNMENT

This section describes the standard data type and boundary alignment. Table 4.4-1 shows the assignment rules.

Boundary Alignment

Table 4.4-1 Variable Assignment Rules

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Assignment Size [Byte]</th>
<th>Boundary Alignment [Byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>signed char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>unsigned int</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>long double</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>near pointer/address</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>far pointer/address</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Structure/union</td>
<td>Explained later</td>
<td>Explained later</td>
</tr>
</tbody>
</table>
4.5 BIT FIELD

This section describes the bit field data size and boundary alignment. The bit field data is assigned to a storage unit that has an adequate size for bit field data retention and is located at the smallest address.

■ Bit Field

Consecutive bit field data are packed at consecutive bits having the same storage unit, without regard to the type, beginning with the LSB and continuing toward the MSB. An example is shown in Figure 4.5-1.

```c
struct tag1 {
    int   A:10;
    short B:3;
    char  C:2;
};
```

Unoccupied | C | B | A
---|---|---|---
15 (MSB) | 13 | 10 | 0 (LSB)

**Figure 4.5-1 Example 1 of Bit Field Data Size and Boundary Alignment**

If a field to be assigned lies over a bit field type boundary, its assignment is completed by aligning it with a boundary suitable for the type. The bit field type boundary of long int type is regarded as 4 and aligned. An example is shown in Figure 4.5-2.

```c
struct tag2 {
    long int   A:12; /* 2-byte boundary data */
    short      B:5; /* 2-byte boundary data */
    char       C:5; /* 2-byte boundary data */
};
```

Unoccupied | C | Unoccupied | B | Unoccupied | A
---|---|---|---|---|---
31 (MSB) | 28 | 24 | 21 | 16 | 12 | 0 (LSB)

**Figure 4.5-2 Example 2 of Bit Field Data Size and Boundary Alignment**

When a bit field having a bit length of 0 is declared, it is forcibly assigned to the next storage unit. An example is shown in Figure 4.5-3.
struct tag3 {
    int A:5;
    int B:5;
    int :0;
    int C:6;
};

Figure 4.5-3 Example 3 of Bit Field Data Size and Boundary Alignment
This section describes the structure/union data size and boundary alignment. The structure/union data size is a multiple of the maximum boundary alignment size of the members. Boundary alignment for the area itself is accomplished by means of member maximum boundary alignment. The individual members are subjected to boundary alignment in accordance with the member type.

**Structure/Union**

Figure 4.6-1 to Figure 4.6-3 show examples concerning structure/union data size and boundary alignment.

![Figure 4.6-1 Example 1 of Structure/Union Data Size and Boundary Alignment](image)

```c
struct st1  { char A; } → sizeof(st1) = 1 BYTE
struct st2  { short A; } → sizeof(st2) = 2 BYTES
struct st3  { char A; short B; } → sizeof(st3) = 4 BYTES
struct st4  { int A; char B; } → sizeof(st4) = 4 BYTES
```

![Figure 4.6-2 Example 2 of Structure/Union Data Size and Boundary Alignment](image)
4.7 FUNCTION CALL INTERFACE (STACK ARGUMENT PASSING)

The general rules for control transfer between functions are established as standard regulations for individual architectures and are called standard linkage regulations. A module written in C language can be combined with a module written using a different method (e.g., assembler language) when the standard linkage regulations are complied with.

- Function Call Interface (stack argument passing)
  - Stack Frame
    The stack frame construction is stipulated by the standard linkage regulations.
  - Argument
    Argument transfer relative to the callee function is effected via a stack and register.
  - Argument Extension Format
    When an argument is to be stored in a stack, the argument type is converted to an extended format in accordance with the argument type.
  - Calling Procedure
    The caller function initiates branching to the callee function after argument storage.
  - Register
    The register guarantee stated in the standard linkage regulations and the register setup regulations are explained later.
  - Return Value
    The return value interface stated in the standard linkage regulations is explained later.
4.7.1 Stack Frame (stack argument passing)

The standard linkage regulations prescribe the stack frame construction.

Stack Frame (stack argument passing)

The stack pointer (SP) always indicates the lowest order of the stack frame. Its address value always represents the work boundary. Figure 4.7-1 shows the standard function stack frame status.

![Figure 4.7-1 Stack Frame (stack argument passing)](image)

(1) Return value address save area

This is the place where the start address of a return value storage area is stored for a function which returns a structure/union/double or long double type.

When a structure/union is the return value, the start address of an area where the caller function stores the return value is stored in accumulator AL and passed to the callee function.

The callee function interprets the address stored in accumulator AL as the storage area start address of the return value.

When the return value address stored in accumulator AL needs to be saved into memory, the callee function saves the address in this return value address save area.

(2) Register save area

This is a register save area that must be guaranteed for the caller function. This area is not secured when the register save operation is not needed.

(3) Local variable save area

This is the area for local variables and temporary variables.

(4) Old FP

This area stores the frame pointer (RW3) value of the caller function.

(5) Return address storage area

This area stores the caller function return address. When a function is called, this area is set up by the caller function.
(6) Argument area/Parameter area

When a function is called, this area is used for argument transfer. When the argument is set up by the caller function, this area is referred to as the argument area. When the argument is referred by the callee function, this area is referred to as the parameter area.

For details, see 4.7.2, “Argument (stack argument passing)”.

(7) Return value area

When a structure, union, double, or long double type return function is called, this area is secured by the caller function. This area does not always have to be secured at this location. However, the callee function performs processing on the assumption that this area is secured in the stack. Therefore, if this area is secured outside the stack, no subsequent operations will be guaranteed.

The compiler secures the double/long double type return function return value area which overlaps the argument area. This is so done as to enhance the object efficiency in some special cases. Therefore, when the double/long double type return function stores the return value in the return value area, it must start with the highest-order address and continue sequentially toward the lowest-order address. Further, a write operation must be conducted after all the parameters are completely referred.
CHAPTER 4 OBJECT PROGRAM STRUCTURE

4.7.2 Argument (stack argument passing)

Argument transfer relative to the callee function is effected via the stack. For an argument less than 2 bytes long or an argument having a size which is not a multiple of 2, an area having a size which is determined by reckoning a less-than-2-byte portion as 2 bytes will be secured within the stack. The argument area is allocated/deallocated by the caller function.

Figure 4.7-2 shows an example of argument transfer relative to the callee function.

```c
struct A { char a; } st;
extern void sub( char, struct A, int );
sub( 1, st, 2 );
```

Figure 4.7-2 Argument Format for Standard Linkage Regulations (stack argument passing)
4.7 FUNCTION CALL INTERFACE (STACK ARGUMENT PASSING)

4.7.3 Argument Extension Format (stack argument passing)

When an argument is to be stored in the stack, its type is converted to an extended type in accordance with the individual argument type. The argument is released by the caller function after the return from the callee function is made.

■ Argument Extension Format (stack argument passing)

Table 4.7-1 shows the argument extension format.

Table 4.7-1 Argument Extension Format (stack argument passing)

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Extended Type*1</th>
<th>Stack Storage Size [Byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>signed char</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>unsigned char</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>short</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>unsigned short</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>unsigned int</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>No extension</td>
<td>4</td>
</tr>
<tr>
<td>unsigned long</td>
<td>No extension</td>
<td>4</td>
</tr>
<tr>
<td>float</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>double</td>
<td>No extension</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>No extension</td>
<td>8</td>
</tr>
<tr>
<td>near pointer/address</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>far pointer/address</td>
<td>No extension</td>
<td>4</td>
</tr>
<tr>
<td>Structure/union</td>
<td>*2</td>
<td>*2</td>
</tr>
</tbody>
</table>

*1: The extended type represents an extended type that is provided when no argument type is given. When a prototype declaration is made, it is complied with. For an argument less than 2 bytes long or an argument having a size which is not a multiple of 2, an area having a size which is determined by reckoning a less-than-2-byte portion as 2 bytes will be secured within the stack even when extension is not effected.

*2: For an argument less than 2 bytes long or an argument having a size which is not a multiple of 2, an area having a size which is determined by reckoning a less-than-2-byte portion as 2 bytes will be secured within the stack.
4.7.4 Calling Procedure (stack argument passing)

The caller function initiates branching to the callee function after argument storage.

- Calling Procedure (stack argument passing)

  Figure 4.7-3 shows the stack frame prevailing at calling in compliance with the standard linkage regulations.

  ![Figure 4.7-3 Stack Frame Prevailing at Calling in Compliance with Standard Linkage Regulations (stack argument passing)](image)

  The callee function saves the caller function frame pointer (RW3) in the stack and then stores the prevailing stack pointer value in the stack as the new frame pointer value. Subsequently, the local variable area and caller register save area are acquired from the stack to save the caller register.

  Figure 4.7-4 shows the stack frame that is created by the callee function in compliance with the standard linkage regulations.

  ![Figure 4.7-4 Stack Frame Created by Callee Function in Compliance with Standard Linkage Regulations (stack argument passing)](image)
4.7 FUNCTION CALL INTERFACE (STACK ARGUMENT PASSING)

4.7.5 Register (stack argument passing)

This section describes the register guarantee and register setup regulations in the standard linkage regulations.

■ Register Guarantee (stack argument passing)

The callee function guarantees the following registers of the caller function.

- General-purpose registers RW0 to RW3, RW6, RW7, and USP (SSP)

The register guarantee is provided when the callee function acquires a new area from the stack and saves the register value in that area. Note, however, that registers remaining unchanged within the function are not saved. If such registers are altered using the asm statement, etc., no subsequent operations will be guaranteed.

■ Register Setup (stack argument passing)

Table 4.7-2 shows the register regulations for function call and return periods.

Table 4.7-2 Register Regulations for Function Call and Return Periods (stack argument passing)

<table>
<thead>
<tr>
<th>Register</th>
<th>Call Period</th>
<th>Return Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Return value area address</td>
<td>Return value*</td>
</tr>
<tr>
<td>RW0 to RW2</td>
<td>Not stipulated</td>
<td>Call period value guaranteed</td>
</tr>
<tr>
<td>RW3</td>
<td>Frame pointer</td>
<td>Call period value guaranteed</td>
</tr>
<tr>
<td>RW4 and RW5</td>
<td>Not stipulated</td>
<td>Not stipulated</td>
</tr>
<tr>
<td>RW6 and RW7</td>
<td>Not stipulated</td>
<td>Call period value guaranteed</td>
</tr>
<tr>
<td>USP (SSP)</td>
<td>Stack pointer</td>
<td>Call period value guaranteed</td>
</tr>
</tbody>
</table>

Note: There are no stipulations for situations where a function without the return value is called or a function having a structure/union/double/long double type return value is called.
4.7.6 Return Value (stack argument passing)

Table 4.7-3 shows the return value interface stated in the standard linkage regulations.

Return Value (stack argument passing)

<table>
<thead>
<tr>
<th>Return Value Type</th>
<th>Return Value Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>None</td>
</tr>
<tr>
<td>char</td>
<td>AL</td>
</tr>
<tr>
<td>signed char</td>
<td>AL</td>
</tr>
<tr>
<td>unsigned char</td>
<td>AL</td>
</tr>
<tr>
<td>short</td>
<td>AL</td>
</tr>
<tr>
<td>unsigned short</td>
<td>AL</td>
</tr>
<tr>
<td>int</td>
<td>AL</td>
</tr>
<tr>
<td>unsigned int</td>
<td>AL</td>
</tr>
<tr>
<td>long</td>
<td>A</td>
</tr>
<tr>
<td>unsigned long</td>
<td>A</td>
</tr>
<tr>
<td>float</td>
<td>A</td>
</tr>
<tr>
<td>near pointer/address</td>
<td>AL</td>
</tr>
<tr>
<td>far pointer/address</td>
<td>A</td>
</tr>
<tr>
<td>double</td>
<td>AL*</td>
</tr>
<tr>
<td>long double</td>
<td>AL*</td>
</tr>
<tr>
<td>Structure/union</td>
<td>AL*</td>
</tr>
</tbody>
</table>

**Note:** The caller function stores the start address of the return value storage area into AL and then passes it to the callee function. The callee function interprets AL as the start address of the return value storage area. When this address needs to be saved in memory, the callee function secures the return value address save area and saves the address in that area.
The general rules for control transfer between functions are established as standard regulations and are called standard linkage regulations. A module written in C language can be combined with a module written using a different method (e.g., assembler language) when the standard linkage regulations are complied with. If changing the function call interface, refer to “Appendix D GUIDE TO CHANGING FUNCTION-CALL INTERFACE”.

- **Function Call Interface (register argument passing)**
  - **Stack Frame**
    The stack frame construction is stipulated by the standard linkage regulations.
  - **Argument**
    Argument transfer relative to the callee function is effected via a stack and register.
  - **Argument Extension Format**
    When an argument is to be stored in a register and a stack, the argument type is converted to an extended format in accordance with the argument type.
  - **Calling Procedure**
    The caller function initiates branching to the callee function after argument storage.
  - **Register**
    The register guarantee stated in the standard linkage regulations and the register setup regulations are explained later.
  - **Return Value**
    The return value interface stated in the standard linkage regulations is explained later.
4.8.1 Stack Frame (register argument passing)

The standard linkage regulations prescribe the stack frame construction.

Stack Frame (register argument passing)

The stack pointer (SP) always indicates the lowest order of the stack frame. Its address value always represents the work boundary. The standard function stack frame status is shown below.

![Stack Frame (register argument passing)](image)

(1) Return value address save area

This is the place where the start address of a return value storage area is stored for a function which returns a structure/union/double or long double type.

When a structure/union is the return value, the start address of a area where the caller function stores the return value is stored in accumulator AL and passed to the callee function.

The callee function interprets the address stored in accumulator AL as the storage area start address of return value.

When the return value address stored in accumulator AL needs to be saved into memory, the callee function saves the address in this return value address save area.

(2) Register save area

This is a register save area that must be guaranteed for the caller function. This area is not secured when the register save operation is not needed.

(3) Local variable save area

This is the area for local variables and temporary variables.

(4) Old FP

This area stores the frame pointer (RW3) value of the caller function.
(5) Argument register save area

The value of argument register saves in this area in the case of following; when using the parameter address passed in the argument registers (RW0, RW1), when not retaining the parameter value passed in the argument register on registers and when optimized level is 0. The argument register save area is not consecutive with the parameter area passed in stacks.

This area is not secured when the register save operation is not needed.

(6) Return address storage area

This area stores the caller function return address. When a function is called, this area is set up by the caller function.

(7) Argument area/Parameter area

When a function is called, this area is used for argument transfer, which is other than the argument passed by parameter register. When the argument is set up by the caller function, this area is referred to as the argument area. When the argument is referred by the callee function, this area is referred to as the parameter area.

For details, see 4.8.2, “Argument (register argument passing)”.  

(8) Return value area

When a structure, union, double, or long double type return function is called, this area is secured by the caller function. This area does not always have to be secured at this location. However, the callee function performs processing on the assumption that this area is secured in the stack. Therefore, if this area is secured outside the stack, no subsequent operations will be guaranteed.

The compiler secures the double/long double type return function return value area which overlaps the argument area. This is so done as to enhance the object efficiency in some special cases. Therefore, when the double/long double type return function stores the return value in the return value area, it must start with the highest-order address and continue sequentially toward the lowest-order address. Further, a write operation must be conducted after all the parameters are completely referred.
CHAPTER 4 OBJECT PROGRAM STRUCTURE

4.8.2 Argument (register argument passing)

Argument transfer relative to the callee function is effected via the argument register or the stack. When the argument is received/sent through the argument register, argument in less than 2 bytes (an argument less than 2 bytes long) is stored in the argument register as value of 2-byte. For an argument having a size which is not a multiple of 2, an area having a size which is determined by reckoning a less-than-2-byte portion as 2 bytes will be secured within the stack. The argument area is allocated/deallocated by the caller function.

■ Argument (register argument passing)

When the function without the prototype declaration of the function is called, the operation is not guaranteed.

All changeable-piece arguments are received/sent through stacks.

Table 4.8-1 shows the method of argument transfer.

**Table 4.8-1 Method of Argument Transfer (register argument passing)**

<table>
<thead>
<tr>
<th>Combination of argument *1</th>
<th>First argument</th>
<th>Second argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>func (H, H)</td>
<td>RW0</td>
<td>RW1</td>
</tr>
<tr>
<td>func (H, W)</td>
<td>RW0</td>
<td>stack</td>
</tr>
<tr>
<td>func (H, O)</td>
<td>RW0</td>
<td>stack</td>
</tr>
<tr>
<td>func (W, H)</td>
<td>RL0 (RW0, RW1)</td>
<td>stack</td>
</tr>
<tr>
<td>func (W, W)</td>
<td>RL0 (RW0, RW1)</td>
<td>stack</td>
</tr>
<tr>
<td>func (W, O)</td>
<td>RL0 (RW0, RW1)</td>
<td>stack</td>
</tr>
<tr>
<td>func (O, H)</td>
<td>stack</td>
<td>stack</td>
</tr>
<tr>
<td>func (O, W)</td>
<td>stack</td>
<td>stack</td>
</tr>
<tr>
<td>func (O, O)</td>
<td>stack</td>
<td>stack</td>
</tr>
</tbody>
</table>

H: Models of less than 2 bytes (char, signed char, unsigned char, short, unsigned short, int, unsigned int, near pointer/address)

W: 4 byte models (long, unsigned long, float with prototype declaration *2, far pointer/address)

O: The other models (float without prototype declaration, double, long double, structure and union)

*1: Since the third argument, any type receives/sends argument through the stack.

*2: The handling of float is different according to the presence of the prototype declaration. It is because of the default argument promotion of C language specification.
The following figure shows the example of receiving/sending argument to callee function. (The example is small model.)

```c
extern int sub (int a, char b, char c);
sub(1,2,3);
```

2 is extended to int type.

Figure 4.8-2 Argument Format for Standard Linkage Regulation (register argument passing)
4.8.3 Argument Extension Format (register argument passing)

When an argument is to be stored in the register or the stack, its type is converted to an extended type in accordance with the individual argument type. The argument passed through stack is released by the caller function after the return from the callee function is made.

Table 4.8-2 shows the argument extension format.

Table 4.8-2 Argument Extension Format (register argument passing)

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Extended Type*1</th>
<th>Register Argument Size or Stack Storage Size [Byte]</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>signed char</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>unsigned char</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>short</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>unsigned short</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>unsigned int</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>No extension</td>
<td>4</td>
</tr>
<tr>
<td>unsigned long</td>
<td>No extension</td>
<td>4</td>
</tr>
<tr>
<td>float</td>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>double</td>
<td>No extension</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>No extension</td>
<td>8</td>
</tr>
<tr>
<td>near pointer/address</td>
<td>No extension</td>
<td>2</td>
</tr>
<tr>
<td>far pointer/address</td>
<td>No extension</td>
<td>4</td>
</tr>
<tr>
<td>Structure/union</td>
<td>*2</td>
<td>*2</td>
</tr>
</tbody>
</table>

*1: The extended type represents an extended type that is provided when it is passed through the argument register and when no argument type passed through the stack is given. For the argument passed through the stack, when a prototype declaration is made, it is complied with. For an argument less than 2 bytes long or an argument having a size which is not a multiple of 2, an area having a size which is determined by reckoning a less-than-2-byte portion as 2 bytes will be secured within the stack even when extension is not effected.

*2: For an argument less than 2 bytes long or an argument having a size which is not a multiple of 2, an area having a size which is determined by reckoning a less-than-2-byte portion as 2 bytes will be secured within the stack.
4.8 FUNCTION CALL INTERFACE (REGISTER ARGUMENT PASSING)

4.8.4 Calling Procedure (register argument passing)

The caller function initiates branching to the callee function after argument storage.

■ Calling Procedure (register argument passing)

Figure 4.8-3 shows the stack frame prevailing at calling in compliance with the standard linkage regulations.

![Stack Frame Diagram](image-url)

Figure 4.8-3 Stack Frame Prevailing at Calling in Compliance with Standard Linkage Regulations (register argument passing)

When the save of the argument register is necessary, the callee function saves register argument to the stack, saves the caller function frame pointer (RW3) in the stack, and then stores the prevailing stack pointer value in the stack as the new frame pointer value. Subsequently, the local variable area and caller register save area are acquired from the stack to save the caller register.
Figure 4.8-4 shows the stack frame that is created by the callee function in compliance with the standard linkage regulations.

<table>
<thead>
<tr>
<th>(Low)</th>
<th>(Callee function) SP →</th>
<th>Return value address save area</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Register save area</td>
<td></td>
</tr>
<tr>
<td>(Callee function) FP →</td>
<td>Local variable area</td>
<td></td>
</tr>
<tr>
<td>(Callee function) old SP →</td>
<td>Old FP</td>
<td></td>
</tr>
<tr>
<td>(Caller function) old SP →</td>
<td>Argument register save area</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parameter area</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return value area</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return value address save area</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Register save area</td>
<td></td>
</tr>
<tr>
<td>(Caller function) old FP →</td>
<td>Local variable area</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 4.8-4 Stack Frame Created by Callee Function in Compliance with Standard Linkage Regulations (register argument passing)*
4.8 FUNCTION CALL INTERFACE (REGISTER ARGUMENT PASSING)

4.8.5 Register (register argument passing)

This section describes the register guarantee in the standard linkage regulations.

- Register Guarantee (register argument passing)
  The callee function guarantees the following registers of the caller function.
  - General-purpose registers RW2, RW3, RW6, RW7, and USP (SSP)
  The register guarantee is provided when the callee function acquires a new area from the stack and saves the register value in that area. Note, however, that registers remaining unchanged within the function are not saved. If such registers are altered using the asm statement, etc., no subsequent operations will be guaranteed.

- Register Setup (register argument passing)
  The register setup in the standard linkage regulations is described.
  Table 4.8-3 shows the register regulations for function call and return periods.

**Table 4.8-3 Register Regulations for Function Call and Return Periods (register argument passing)**

<table>
<thead>
<tr>
<th>Register</th>
<th>Call Period</th>
<th>Return Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Return value area address</td>
<td>Return value*</td>
</tr>
<tr>
<td>RW0, RW1</td>
<td>Argument register</td>
<td>Not stipulated</td>
</tr>
<tr>
<td>RW2</td>
<td>Not stipulated</td>
<td>Call period value guaranteed</td>
</tr>
<tr>
<td>RW3</td>
<td>Frame pointer</td>
<td>Call period value guaranteed</td>
</tr>
<tr>
<td>RW4, WR5</td>
<td>Not stipulated</td>
<td>Not stipulated</td>
</tr>
<tr>
<td>RW6, RW7</td>
<td>Not stipulated</td>
<td>Call period value guaranteed</td>
</tr>
<tr>
<td>USP (SSP)</td>
<td>Stack pointer</td>
<td>Call period value guaranteed</td>
</tr>
</tbody>
</table>

**Note:** There are no stipulations for situations where a function without the return value is called or a function having a structure/union/double/long double type return value is called.
4.8.6 Return Value (register argument passing)

Table 4.8-4 shows the return value interface stated in the standard linkage regulations.

<table>
<thead>
<tr>
<th>Argument Type</th>
<th>Return Value Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>None</td>
</tr>
<tr>
<td>char</td>
<td>AL</td>
</tr>
<tr>
<td>signed char</td>
<td>AL</td>
</tr>
<tr>
<td>unsigned char</td>
<td>AL</td>
</tr>
<tr>
<td>short</td>
<td>AL</td>
</tr>
<tr>
<td>unsigned short</td>
<td>AL</td>
</tr>
<tr>
<td>int</td>
<td>AL</td>
</tr>
<tr>
<td>unsigned int</td>
<td>AL</td>
</tr>
<tr>
<td>long</td>
<td>A</td>
</tr>
<tr>
<td>unsigned long</td>
<td>A</td>
</tr>
<tr>
<td>float</td>
<td>A</td>
</tr>
<tr>
<td>near pointer/address</td>
<td>AL</td>
</tr>
<tr>
<td>far pointer/address</td>
<td>A</td>
</tr>
<tr>
<td>double</td>
<td>AL*</td>
</tr>
<tr>
<td>long double</td>
<td>AL*</td>
</tr>
<tr>
<td>Structure/union</td>
<td>AL*</td>
</tr>
</tbody>
</table>

Note: The caller function stores the start address of the return value storage area into AL and then passes it to the callee function. The callee function interprets AL as the start address of the return value storage area. When this address needs to be saved in memory, the callee function secures the return value address save area and saves the address in that area.
4.9 INTERRUPT FUNCTION CALL INTERFACE

The interrupt function can be written using the `__interrupt` type qualifier. If the interrupt function is called by a method other than an interrupt, no subsequent operations will be guaranteed. The function call interface within the interrupt function is the same as stated in the standard linkage regulations.

- **Interrupt Function Call Interface**
  - **Interrupt Stack Frame**
    When an interrupt occurs, the stack is changed to the interrupt stack.
  - **Argument**
    No argument can be specified for the interrupt function. If any argument is specified for the interrupt function, no subsequent operations will be guaranteed.
  - **Interrupt Function Calling Procedure**
    The interrupt function is called by an interrupt via the interrupt vector table. If the interrupt function is called by any other method, no subsequent operations will be guaranteed.
  - **Register**
    As regards the interrupt function, all registers are guaranteed.
  - **Return Value**
    The interrupt function does not usually have a return value.
4.9.1 Interrupt Stack Frame

When an interrupt occurs, the stack is changed to the interrupt stack.

Interrupt Stack Frame

When an interrupt occurs, the stack pointer (USP) is replaced by the interrupt stack pointer (SSP). Within the interrupt function, the interrupt stack pointer is used as the normal stack pointer.

Figure 4.9-1 shows the interrupt stack frame status prevailing immediately after interrupt generation.

---

Figure 4.9-1 Interrupt Stack Frame
4.9.2 Interrupt Function Calling Procedure

The interrupt function is called by an interrupt via the interrupt vector table. If the interrupt function is called by any other method, no subsequent operations will be guaranteed.

Figure 4.9-2 shows an example of interrupt vector table.
CHAPTER 5    EXTENDED LANGUAGE SPECIFICATIONS

This chapter describes the extended language specifications supported by the compiler and the limitations on compiler translation.

5.1 ASSEMBLER DESCRIPTION FUNCTIONS
5.2 INTERRUPT CONTROL FUNCTIONS
5.3 I/O AREA ACCESS FUNCTION
5.4 direct AREA ACCESS FUNCTION
5.5 16-BIT/24-BIT ADDRESSING ACCESS FUNCTION
5.6 IN-LINE EXPANSION SPECIFYING FUNCTION
5.7 SECTION NAME CHANGE FUNCTION
5.8 REGISTER BANK NUMBER SETUP FUNCTION
5.9 INTERRUPT LEVEL SETUP FUNCTION
5.10 SYSTEM STACK USE SPECIFYING FUNCTION
5.11 STACK BANK AUTOMATIC DISTINCTION FUNCTION
5.12 NO-REGISTER-SAVE INTERRUPT FUNC. FUNCTION
5.13 BUILT-IN FUNCTION
5.14 PREDEFINED MACROS
5.15 LIMITATIONS ON COMPILER TRANSLATION
5.1 ASSEMBLER DESCRIPTION FUNCTIONS

There are the following two assembler description functions.

- **asm statement**
- **Pragma instruction**

### Description by asm Statement

When the `asm` statement is written, the character string literal is expanded as the assembler instruction. This function makes it possible to write the `asm` statement inside and outside the function.

**[General Format]**

```
__asm (Character string literal);
```

**[Explanation]**

- When the `asm` statement is written inside the function, the assembler is expanded at the written position.
- When the statement is written outside the function, it is expanded as an independent section. Therefore, if the statement is to be written outside the function, be sure to write the section definition pseudo instruction to define the section. If the section is not defined, no subsequent operations will be guaranteed.
- When using a general-purpose register within the `asm` statement in the function, the user is responsible for register saving and restoration. The accumulator can be freely used.
- If the `asm` statement exists in a C source program, various optimization features are inhibited even when the `-O` optimization option is specified.

**[Output Example]**

- **Input:**

```
/* When written inside the function */
extern int temp;
sample(){
    __asm("    MOVN    A, #1");
    __asm("    MOVWMOV _temp, A");
}

/* When written outside the function */
__asm("       .SECTION        DATA, DATA, ALIGN=2");
__asm("       .ALIGN  2");
__asm("       .GLOBAL _a");
__asm("   _a:  .RES.B  2");
```

- **Output:**

```
.SECTION        CODE, CODE, ALIGN=1
;-------begin_of_function
.GLOBAL _sample
```
5.1 ASSEMBLER DESCRIPTION FUNCTIONS

```assembly
_sample:
    LINK   #0
    MOVN   A, #1
    MOVW   _temp, A
    UNLINK
    RET

.SECTION        DATA, DATA, ALIGN=2
    .ALIGN  2
    .GLOBAL _a

_a:     .RES.B  2
```

■ Description by Pragma Instruction

The description between #pragma asm and #pragma endasm is directly expanded as the assembler instruction. This function makes it possible to write the statement inside and outside the function.

[General Format]

```
#pragma asm
    Assembler description
#pragma endasm
```

[Explanation]

When the statement is written inside the function, the assembler is expanded at the written position.

When the statement is written outside the function, it is expanded as an independent section. Therefore, if the statement is to be written outside the function, be sure to write the section definition pseudo instruction to define the section. If the section is not defined, no subsequent operations will be guaranteed.

When using a general-purpose register within the asm statement in the function, the user is responsible for register saving and restoration. The accumulator can be freely used.

If the assembler provided by #pragma asm/endasm exists in the C source program, various optimization features are inhibited even when the -O optimization option is specified.
[Output Example]

- **Input:**
  
  ```c
  /* When written inside the function */
  sample()
  #pragma asm
    MOVN A, #1
    MOVW _temp, A
  #pragma endasm
  }
  /* When written outside the function */
  #pragma asm
    .SECTION CODE, CODE, ALIGN=1
    .ALIGN 2
    .GLOBAL _a
    _a: .RES.B 2
  #pragma endasm
  ``

- **Output:**
  
  ```assembly
  .SECTION CODE, CODE, ALIGN=1
  ;-------begin_of_function
  .GLOBAL _sample
  _sample:
    LINK #0
    MOVN A, #1
    MOVW _temp, A
    UNLINK
    RET
  .SECTION DATA, DATA, ALIGN=2
  .ALIGN 2
  .GLOBAL _a
  _a: .RES.B 2
  ```
5.2 INTERRUPT CONTROL FUNCTIONS

There are the following five interrupt control functions.

- Interrupt mask setup function
- Interrupt mask disable function
- Interrupt level setup function
- Interrupt function description function
- Interrupt vector table generation function

**Interrupt Mask Setup Function**

[General Format]
```c
void __DI(void);
```

[Explanation]
Expands the interrupt masking code

[Output Example]
- Input:
  ```c
  __DI();
  ```
- Output:
  ```c
  AND CCR, #191
  ```

**Interrupt Mask Disable Function**

[General Format]
```c
void __EI(void);
```

[Explanation]
Expands the interrupt masking disable code

[Output Example]
- Input:
  ```c
  __EI();
  ```
- Output:
  ```c
  OR CCR, #64
  ```

**Interrupt Level Setup Function**

[General Format]
```c
void __set_il(int level);
```

[Explanation]
Expands the code for changing the interrupt level to level

[Output Example]
- Input:
  ```c
  __set_il(2);
  ```
- Output:
  ```c
  MOV ILM, #2
  ```
CHAPTER 5 EXTENDED LANGUAGE SPECIFICATIONS

■ Interrupt Function Description Function

[General Format 1]

__interrupt void  Interrupt function (void) { ... }

[General Format 2]

extern __interrupt void  Interrupt function (void);

[Explanation]

The interrupt function can be written by specifying the __interrupt type qualifier. Since the interrupt function is called by an interrupt, it is impossible to set up an argument or obtain a return value.

If a function declared or defined by the __interrupt type qualifier is called by performing the normal function calling procedure, no subsequent operations will be guaranteed.

[Output Example]

• Input:

__interrupt void sample(void){ ... }

• Output:

_sample:

LINK   #0
....
UNLINK
RETI

■ Interrupt Vector Table Generation Function

[General Format]

#pragma intvect  Interrupt function name  Vector number  [Mode value]
#pragma defvect  Interrupt function name

[Explanation]

#pragma intvect generates an interrupt vector table for which the interrupt function is set.
#pragma defvect specifies the default interrupt function to be set for interrupt vectors not specified by #pragma intvect.

The interrupt vector table is generated in an independent section named INTVECT.

When #pragma defvect is written, tables for all vectors are generated. Therefore, all vector tables must be defined using the same translation unit. If #pragma defvect is not used, #pragma intvect can be written using two or more translation units.

However, change the section name of the vector table for each file so that the same section name is not outputted.

The definition cannot be formulated two or more times for the same vector number. However, no error occurs if the definitions are for the same translation unit and are identical.

No value other than an integer constant may be specified as the vector number. Specify a vector number between 0 and 255.

No value other than an integer constant may be specified as the mode value.
5.3 I/O AREA ACCESS FUNCTION

The I/O area operation variable can be defined by specifying the __io type qualifier.

---

### I/O Area Access Function

**[General Format]**

```
__io  Variable definition;
```

**[Explanation]**

A variable operating an I/O area defined at addresses between 0x00 and 0xff can be defined by specifying the __io type qualifier.

Since a highly-efficient dedicated instruction is provided for I/O area access, a higher-speed, more-compact object can be generated. This instruction cannot be used for variables operating an I/O area positioned at addresses higher than 0xff. To define a variable that accesses such an area, use the volatile type qualifier.

The initial value cannot be specified for variables for which the __io type qualifier is specified.

When the specified variable is for a structure or union, it is assumed that all members are positioned in the I/O area. The variable cannot be specified for structure or union members. For the variable for which the __io type qualifier is specified, compilation is conducted on the assumption that the volatile type qualifier is specified.

When the -K NOVOLATILE option is specified, the volatile type qualifier is not assumed to be specified for the variable for which the __io type qualifier is specified.

**[Output Example]**

- **Input:**
  
  ```
  #pragma section IO=IOA,attr=IO,locate=0x10
  __io int a;
  void func(void){ a=1;}
  ```

- **Output:**
  
  ```
  .SECTION        IOA, IO, LOCATE=H'0:H'10
  .ALIGN  2
  .GLOBAL _a
  _a:
  .RES.B  2
  .SECTION        CODE, CODE, ALIGN=1
  ;-------begin_of_function
  .GLOBAL _func
  _func:
  LINK   #0
  MOVN   A, #1
  MOVW   I:_a, A
  UNLINK
  RET
  ```
5.4 direct AREA ACCESS FUNCTION

The direct area operation variable can be defined by specifying the __direct type qualifier.

- direct Area Access Function
  [General Format]
  __direct  Variable definition;
  [Explanation]
  The direct area operation variable can be defined by specifying the __direct type qualifier. It makes it possible to specify that the pointer-specified object is the direct area.
  When the specified variable for a structure or union, it is assumed that all members are positioned in the direct area. The variable cannot be specified for structure or union members.
  Since highly-efficient dedicated instructions are provided for direct area accessing, compact objection generation can be achieved at an increased speed.
  In the fcc907s command, to make accessible the section (DIRDATA/DIRINIT) generated by __direct type qualifying the variable, it is necessary to properly set up the DPR with the startup routine.

< Caution >
The compiler generates DIRDATA and DIRINT sections when __direct type qualifier is used. These sections need to be placed consecutively and the united areas cannot be arranged by stepping over 256 byte boundary.
Arrange them on linking with the following specification.

-ra RAM_DIR=DIRarea_start_address/end_address
-sc DIRDATA+DIRINT=RAM_DIR

With the above condition, the following warning will be generated on linking if the size of the united areas is over 256byte.

***W1377L: Total size of the DIR attribute section exceeded 256 bytes

When the source is written in assembly code and 2 or more direct pages are used, the same specification is required for each page. When different page is used for each task of Realtime OS, the same specification is required for each page.
5.4 direct AREA ACCESS FUNCTION

[Output Example]

- Input:
  ```
  int __direct p;
  void sample(void){ p=1;}
  ```

- Output:
  ```
  .SECTION    DIRDATA, DIR, ALIGN=2
  .ALIGN  2
  .GLOBAL _p
  _p:
  .RES.B  2
  .GLOBAL LOADSPB
  .SECTION    CODE, CODE, ALIGN=1
  \;-------begin_of_function
  .GLOBAL _sample
  _sample:
  LINK   #0
  MOVN   A, #1
  MOVW   S:_p, A
  UNLINK
  RET
  ```
5.5  16-BIT/24-BIT ADDRESSING ACCESS FUNCTION

The address space where variables are positioned can be specified by specifying the __near/__far type qualifier. A highly efficient program can be generated by specifying an appropriate address space.

■ 16-bit/24-bit Addressing Access Function

[General Format]

__near  Variable definition;
__far   Variable definition;

[Explanation]

The variable arrangement address space can be specified by specifying the __near/__far type qualifier.

When the __near type qualifier is specified, variables can be positioned in the 16-bit address space.

When the __far type qualifier is specified, variables can be positioned in the 24-bit address space.

A highly efficient program can be generated by specifying an appropriate address space.

If the __near/__far type qualifier is omitted, the address space specified by the memory model employed at the time of compilation is used as the default choice.

The local variable cannot be qualified.

When the far pointer is type-converted to the near pointer, the eight high-order bits are discarded.

When the near pointer is type-converted to the far pointer, the DTB value is used for the eight high-order bits.

When the local variable address is stored in the far pointer, the USB (or SSB) value is used for the eight high-order bits. However, if the local variable address is stored in the far pointer after it has been substituted (or cast) into the near pointer, the DTB value is used so that erratic operations may result.

When a __near type qualified function is to be called from a __far type qualified function, both functions must be positioned in the same section. The reason is that the PCB set up for __far type qualified function calling is used as for __near type qualified function calling.

Variables have to be adjusted to the bank boundary. If not, the generated code cannot access such variable correctly.

[Output Example]

• Input:

    int __near p;
    int __far q;
    void sample(void){ p=1; q=2;}
5.5 16-BIT/24-BIT ADDRESSING ACCESS FUNCTION

- Output:

```
.SECTION DATA_e, DATA, ALIGN=2
FAR_DATA_S:
    .ALIGN 2
    .GLOBAL _q
_q:
    .RES.B 2
.SECTION DATA, DATA, ALIGN=2
    .ALIGN 2
    .GLOBAL _p
_p:
    .RES.B 2
.SECTION DATA_e, DATA, ALIGN=2
FAR_DATA_E:
    .SECTION CODE, CODE, ALIGN=1
;--------begin_of_function
    .GLOBAL _sample
_sample:
    LINK #0
    MOVN A, #1
    MOVW _p, A
    MOV A, #bnksym_q
    MOV ADB, A
    MOVN A, #2
    MOVW ADB:_q, A
    UNLINK
    RET
```
This function specifies the user definition function for in-line expansion. In-line expansion can be specified with the -x option.

**In-line Expansion Specifying Function**

[General Format]

```c
#pragma inline Function name [, Function name ...]
```

[Explanation]

Recursively called functions cannot be subjected to in-line expansion. It should also be noted that functions may not be subjected to in-line expansion depending on asm statement use, structure/union type argument presence or absence, setjmp function calling, and other conditions.

When there are two or more descriptions for the same translation unit or in-line expansion is specified by an option, all the specified function names are valid.

The in-line expansion specifying is invalid if the -O option is not specified.
5.7 SECTION NAME CHANGE FUNCTION

This function is used to change the section name or section attribute and sets the section arrangement address.

■ Section Name Change Function

[General Format]

#pragma section DEFSECT[=NEWNAME][,attr=SECTATTR][,locate=ADDR]

[Explanation]

The section name output by the compiler is changed from DEFSECT to NEWNAME and the section type is changed to SECTATTR. (Please do not describe the blank before and behind =.)

Large, compact and medium models, __far-type qualified variables, and functions can be assigned a section name by prefixing them with FAR_.

It is also possible to select an arrangement address of ADDR.

For the section name output by the compiler, see 4.1, “SECTION STRUCTURE”. For the section type, refer to the Assembler Manual.

When an arrangement address is given, it cannot be specified for the section at linking.

This feature can be specified only once for the same section. When specifying it two or more times, only the last specification is effective.

When the same section name is changed by -s option, only specification by the option is effective.

<Caution>

The operation is not guaranteed when the section having the location address is specified and the section having the location address is not specified exist together the same section name.

[Output Example]

• Input:
  #pragma section CODE=program,attr=CODE,locate=0xff
  void main(void){}

• Output:
  .SECTION  program, CODE, LOCATE=H'0:H'FF
  ;-------begin_of_function
  .GLOBAL _main
  _main:
  LINK  #0
  UNLINK
  RET
CHAPTER 5 EXTENDED LANGUAGE SPECIFICATIONS

[General Format]

```
#pragma  segment DEFSECT[=NEWNAME][,attr=SECTATTR][,locate=ADDR]
```

[Explanation]

The section name output by the compiler is changed from `DEFSECT` to `NEWNAME` and the section type is changed to `SECTATTR`. (Please do not describe the blank before and behind =.)

There are some differences concerning the description about the general format though it is the same as the `#pragma section`.

The `#pragma segment` can be described the plural in the file and acts on the function or the variable defined since the described line. This specification is effective until the `#pragma segment` of same next `DEFSECT` is described. (The description of the `#pragma segment` that `DEFSECT` is different does not influence mutually.)

When `#pragma segment` without `NEWNAME` is described, the section name of `DEFSECT` since the line becomes the section name of default.

When neither the function nor the variable on which it acts since the line where the `#pragma segment` is described are defined, the `#pragma segment` is disregarded.

The `#pragma section` and `-s` option of the section alone not specified by the `#pragma segment` act when the `#pragma segment`, the `#pragma section` or `-s` option is specified at the same time.

The `INTVECT`, `DTRANS` and `DCLEAR` section cannot specify the `#pragma segment`. When `INIT` or `FAR_INIT` and `DCONST` or `FAR_DCONST` are specified for `DEFSECT`, it is necessary to specify corresponding `DCONST` or `FAR_DCONST` and `INIT` or `FAR_INIT`. Moreover, it is necessary to transfer an initial value in the startup routine.

[Output Example]

- **Input:**

```
#pragma segment CODE=program1
void func1(void){}
#pragma segment DATA=ram1
int a1;
#pragma segment CODE=program2
void func2(void){}
#pragma segment DATA=ram2
int a2;
```

- **Output:**

```
.SECTION ram2, DATA, ALIGN=2
.ALIGN 2
.GLOBAL _a2
_a2:
.RES.B 2
.SECTION ram1, DATA, ALIGN=2
.ALIGN 2
.GLOBAL _a1
```
5.7 SECTION NAME CHANGE FUNCTION

_a1:

.RES.B 2

.SECION program1, CODE, ALIGN=1

.GLOBAL _func1

_func1:

RET

.SECION program2, CODE, ALIGN=1

.GLOBAL _func2

_func2:

RET

<Caution>

#pragma segment works on the position of the first variable definition/variable declaration in the file.

Please direct the variable declaration the change in the section name if there is a variable declaration before the variable definition.

#pragma segment works on the position where the function is defined in the file if the target is a function. The function declaration before the position where the function is defined does not influence the section name.

The operation is not guaranteed when the section having the location address is specified and the section having the location address is not specified exist together the same section name.

[Output Example]

• Input:

#pragma segment CONST=const1,attr=CONST,locate=0xff00
extern const int var; //Variable declaration
#pragma segment CONST=const2,attr=CONST,locate=0xff10
const int var=10; //Variable definition
#pragma segment CODE=program1,attr=CODE,locate=0xff20
extern void func(void); //function declaration
#pragma segment CODE=program2,attr=CODE,locate=0xff30
void func(void)(); //function definition

• Output section of variable/function:

<table>
<thead>
<tr>
<th>Variable/function names</th>
<th>Section name</th>
</tr>
</thead>
<tbody>
<tr>
<td>var</td>
<td>const1</td>
</tr>
<tr>
<td>func</td>
<td>program2</td>
</tr>
</tbody>
</table>
This function is used to specify the register bank that the function uses.

### Register Bank Number Setup Function

**[General Format]**

```c
#pragma register(NUM)
#pragma noregister
```

**[Explanation]**

#pragma register specifies the register bank that the subsequently-defined function uses.

#pragma noregister clears the register bank specifying.

An integer constant between 0 and 31 can be specified in the NUM position to specify the register bank number. A hexadecimal, octal, or decimal number can be described.

Although the register bank number is changed at the beginning of the specified function, remember that the new number does not revert to the previous number at completion of function execution (the case of the interrupt function is excluded).

Always specify #pragma register and #pragma noregister as a set. Nesting is not possible.

**[Output Example]**

- **Input:**
  ```c
  #pragma register(2)
  void func(void){}
  #pragma noregister
  ```

- **Output:**
  ```c
  _func:
  MOV     RP, #2
  LINK    #0
  UNLINK
  RET
  ```
5.9 INTERRUPT LEVEL SETUP FUNCTION

This function is used to set the function interrupt level.

■ Interrupt Level Setup Function

[General Format]

```c
#pragma ilm(NUM)
#pragma noilm
```

[Explanation]

#pragma ilm specifies the interrupt level for the subsequently defined function.
#pragma noilm clears the interrupt level specifying.
The integer constants 0 to 7 can be specified as NUM.
A hexadecimal, octal, or decimal number can be described.
Although the interrupt level is changed at the beginning of the specified function, remember that the new interrupt level does not revert to the previous level at completion of function execution.

Always specify #pragma ilm and #pragma noilm as a set. Nesting is not possible.

[Output Example]

• Input:

```c
#pragma ilm(1)
void func(void){}
#pragma noilm
```

• Output:

```c
_func:
    MOV    ILM, #1
    LINK   #0
    UNLINK
    RET
```
5.10 SYSTEM STACK USE SPECIFYING FUNCTION

This function is used to notify the compiler that the system stack is used by the function.

### System Stack Use Specifying Function

[General Format]

```plaintext
#pragma ssb
#pragma nossb
```

[Explanation]

- `#pragma ssb` notifies the compiler that the system stack is used by the subsequently-defined function.
- `#pragma nossb` clears such a specifying.

Always specify `#pragma ssb` and `#pragma nossb` as a set. Nesting is not possible. `#pragma ssb` cannot be written between `#pragma except` and `#pragma noexcept`.

[Output Example]

**Input:**

```plaintext
__far int *p;
#pragma ssb
void func(void){
  int a;
  p=&a;
}
#pragma nossb
```

**Output:**

```plaintext
_func:
LINK    #2
MOV     A, SSB
MOVEA   A, @RW3+-2
MOVL    _p, A
UNLINK
RET
```
5.11 STACK BANK AUTOMATIC DISTINCTION FUNCTION

This function is used to notify the compiler that the function is operative in both the system stack and user stack.

- Stack Bank Automatic Distinction Function

  [General Format]
  
  ```plaintext
  #pragma except
  #pragma noexcept
  ```

  [Explanation]
  
  #pragma except notifies the compiler that the subsequently-defined function is operative in both the system stack and user stack.
  
  #pragma noexcept clears such a specifying.
  
  Always specify #pragma except and #pragma noexcept as a set. Nesting is not possible.
  
  #pragma except cannot be written between #pragma ssb and #pragma nossb.

  [Output Example]
  
  - Input:
    ```plaintext
    __far int *p;
    #pragma except
    void func(void){
      int a;
      p=&a;
    }
    #pragma noexcept
    ```
  
  - Output:
    ```plaintext
    _func:
    LINK   #2
    CALLP  LOADSPB
    MOVEA  A, @RW3+2
    MOVL   __p, A
    UNLINK
    RET
    ```
5.12 NO-REGISTER-SAVE INTERRUPT FUNC. FUNCTION

This function is used to specify "no function saving".

■ No-register-save Interrupt Func. Function

[General Format]

__nosavereg Function definition

[Explanation]

The __nosavereg type qualifier can be specified to define a function that is not to be saved to a register. This function is used to inhibit the register save operation when it is not needed due to register bank switching.

Register bank switching can be performed using #pragma register. #pragma register is usually used with __interrupt.

[Output Example]

• Input:
  extern void sub(void);
  #pragma register(5)
  __nosavereg __interrupt void func(void){sub();}
  #pragma noregister

• Output:
  __func:
    MOV     RP, #5
    LINK    #0
    CALL    _sub
    UNLINK
    RETI
5.13 BUILT-IN FUNCTION

The following built-in functions are available.

- __wait_nop
- __mul
- __div
- __mod
- __mulu
- __divu
- __modu

### __wait_nop Built-in Function

[General Format]

```c
void __wait_nop(void);
```

[Explanation]

To properly timing of I/O access and interrupt generation, formerly, the NOP instruction was inserted using the asm statement. However, when such a method is used, the asm statement may occasionally inhibit various forms of optimization and greatly degrade the file object efficiency.

When the __wait_nop() built-in function is written, the compiler outputs one NOP instruction to the function call entry position. If the function call entry is performed a count of times until all the issued NOP instructions are covered, timing control is exercised to minimize the effect on optimization.

[Output Example]

- Input:
  ```c
  void sample(void){__wait_nop();}
  ```

- Output:
  ```c
  _sample:
  LINK  #0
  NOP
  UNLINK
  RET
  ```
CHAPTER 5 EXTENDED LANGUAGE SPECIFICATIONS

■ __mul Built-in Function

[General Format]

    signed long __mul(signed int, signed int);

[Explanation]

This function multiplies signed 16-bit data by signed 16-bit data to return a signed 32-bit result.

It is possible to avert a 16-bit computation-induced overflow by using this built-in function, thereby increasing computation efficiency.

It expands only when the F^2MC-16LX/16F family MB number is specified as the -cpu option. However, this function is not expanded in the MB90500 series when -div905 option is not specified.

[Output Example]

• Input:

    extern signed int arg1, arg2;
    extern signed long ans;
    void sample(void){
        ans = __mul(arg1, arg2);
    }

• Output:

    MOVW  A, _arg1
    MULW  A, _arg2
    MOVL  _ans, A
5.13 BUILT-IN FUNCTION

■ __div Built-in Function

[General Format]

```c
signed int __div(signed long, signed int);
```

[Explanation]

This function performs a division between signed 32-bit data and signed 16-bit data to return a signed 16-bit result.

It is possible to achieve increased computation efficiency by using this built-in function.

It expands only when the F²MC-16LX/16F family MB number is specified as the -cpu option. However, this function is not expanded in the MB90500 series when -div905 option is not specified.

[Output Example]

- Input:
  ```c
  extern signed int arg2, ans;
  extern signed long arg1;
  void sample(void){
    ans = __div(arg1, arg2);
  }
  ```

- Output:
  ```
  MOVL    A, _arg1
  MOVW    RW0, _arg2
  DIVW    A, RW0
  MOVW    _ans, A
  ```

■ __mod Built-in Function

[General Format]

```c
signed int __mod(signed long, signed int);
```

[Explanation]

This function performs a modulo operation between signed 32-bit data and signed 16-bit data to return a signed 16-bit result.

It is possible to achieve increased computation efficiency by using this built-in function.

It expands only when the F²MC-16LX/16F family MB number is specified as the -cpu option. However, this function is not expanded in the MB90500 series when -div905 option is not specified.

[Output Example]

- Input:
  ```c
  extern signed int arg2, ans;
  extern signed long arg1;
  void sample(void){
    ans = __mod(arg1, arg2);
  }
  ```
CHAPTER 5 EXTENDED LANGUAGE SPECIFICATIONS

• Output:
   
   ```
   MOVL    A, _arg1
   MOVW    RW0, _arg2
   MODW    RW0
   MOVW    A, RW0
   MOVW    _ans, A
   ```

■ **__mulu** Built-in Function

[General Format]

```
unsigned long __mulu(unsigned int, unsigned int);
```  

[Explanation]

This function multiplies unsigned 16-bit data by unsigned 16-bit data to return an unsigned 32-bit result.

It is possible to avert a 16-bit computation-induced overflow by using this built-in function, thereby increasing computation efficiency.

[Output Example]

• Input:

   ```
   extern unsigned int arg1, arg2;
   extern unsigned long ans;
   void sample(void){
       ans = __mulu(arg1, arg2);
   }
   ```

• Output:

   ```
   MOVW    A, _arg1
   MULUW   A, _arg2
   MOVL    _ans, A
   ```

■ **__divu** Built-in Function

[General Format]

```
unsigned int __divu(unsigned long, unsigned int);
```  

[Explanation]

This function performs a division between unsigned 32-bit data and unsigned 16-bit data to return an unsigned 16-bit result.

It is possible to achieve increased computation efficiency by using this built-in function.
[Output Example]

- Input:
  ```c
  extern unsigned int arg2, ans;
  extern unsigned long arg1;
  void sample(void){
    ans = __divu(arg1, arg2);
  }
  ```

- Output:
  ```assembly
  MOVL    A, _arg1
  MOVW    RW0, _arg2
  DIVUW   A, RW0
  MOVW    _ans, A
  ```

■ __modu Built-in Function

[General Format]

```c
unsigned int __modu(unsigned long, unsigned int);
```

[Explanation]

This function performs a modulo operation between unsigned 32-bit data and unsigned 16-bit data to return an unsigned 16-bit result.

It is possible to achieve increased computation efficiency by using this built-in function.

[Output Example]

- Input:
  ```c
  extern unsigned int arg2, ans;
  extern unsigned long arg1;
  void sample(void){
    ans = __modu(arg1, arg2);
  }
  ```

- Output:
  ```assembly
  MOVL    A, _arg1
  MOVW    RW0, _arg2
  MODUW   RW0
  MOVW    A, RW0
  MOVW    _ans, A
  ```
5.14 PREDEFINED MACROS

This section describes the macro names predefined by the compiler.

- Macros Stipulated by ANSI Standard

  The ANSI standard stipulates the following macros.

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LINE</strong></td>
<td>Defines line number of current source line</td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>Defines source file name</td>
</tr>
<tr>
<td><strong>DATE</strong></td>
<td>Defines source file translation date</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>Defines source file translation time</td>
</tr>
<tr>
<td><strong>STDC</strong></td>
<td>Macro indicating that the processing system meets requirements. When the -Ja option is specified, 0 is selected as the definition. When the -Jc option is specified, 1 is selected as the definition.</td>
</tr>
</tbody>
</table>

- Macros Predefined by fcc907s Command

  The fcc907s command predefined the following macros.

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>COMPILER_FCC907</strong></td>
<td>Selects 1 as definition</td>
</tr>
<tr>
<td><strong>CPU_MB number</strong></td>
<td>Selects MB number specified by the -cpu option as definition</td>
</tr>
<tr>
<td><strong>CPU_16L</strong></td>
<td>Selects 1 as definition for macro of certain series name in accordance with MB number specified by the -cpu option</td>
</tr>
<tr>
<td><strong>CPU_16LX</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CPU_16F</strong></td>
<td></td>
</tr>
</tbody>
</table>
Table 5.15-1 shows the translation limitations to be imposed when the compiler is used. The table also indicates the minimum ANSI standard to be met.

### Limitations on Compiler Translation

#### Table 5.15-1 List of Translation Limitations

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
<th>ANSI Standard</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Count of nesting levels for a compound statement, repetition control structure, and selection control structure</td>
<td>15</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>Count of nesting levels for condition incorporation</td>
<td>8</td>
<td>∞</td>
</tr>
<tr>
<td>3</td>
<td>Count of pointers, arrays, and function declarators (any combinations of these) for qualifying one arithmetic type, structure type, union type, or incomplete type in a declaration</td>
<td>12</td>
<td>∞</td>
</tr>
<tr>
<td>4</td>
<td>Count of nests provided by parentheses for one complete declarator</td>
<td>31</td>
<td>∞</td>
</tr>
<tr>
<td>5</td>
<td>Count of nest expressions provided by parentheses for one complete expression</td>
<td>32</td>
<td>∞</td>
</tr>
<tr>
<td>6</td>
<td>Count of valid leading characters of internal identifier or macro name</td>
<td>31</td>
<td>∞</td>
</tr>
<tr>
<td>7</td>
<td>Count of valid leading characters of external identifier</td>
<td>6</td>
<td>254*</td>
</tr>
<tr>
<td>8</td>
<td>Count of external identifiers of one translation unit</td>
<td>511</td>
<td>∞</td>
</tr>
<tr>
<td>9</td>
<td>Count of identifiers having the block valid range in one block</td>
<td>127</td>
<td>∞</td>
</tr>
<tr>
<td>10</td>
<td>Count of macro names that can be simultaneously defined by one translation unit</td>
<td>1024</td>
<td>∞</td>
</tr>
<tr>
<td>11</td>
<td>Count of parameters in one function definition</td>
<td>31</td>
<td>∞</td>
</tr>
<tr>
<td>12</td>
<td>Count of arguments for one function call</td>
<td>31</td>
<td>∞</td>
</tr>
<tr>
<td>13</td>
<td>Count of parameters in one macro definition</td>
<td>31</td>
<td>∞</td>
</tr>
<tr>
<td>14</td>
<td>Count of arguments in one macro call</td>
<td>31</td>
<td>∞</td>
</tr>
<tr>
<td>15</td>
<td>Maximum count of characters in one logical source line</td>
<td>509</td>
<td>∞</td>
</tr>
<tr>
<td>16</td>
<td>Count of characters in a (linked) byte character string literal or wide-angle character string literal (terminal character included)</td>
<td>509</td>
<td>∞</td>
</tr>
<tr>
<td>17</td>
<td>Count of bytes for one arithmetic unit</td>
<td>32767</td>
<td>65535</td>
</tr>
<tr>
<td>18</td>
<td>Count of nesting levels for #include file</td>
<td>8</td>
<td>252</td>
</tr>
<tr>
<td>19</td>
<td>Count of case name cards in one switch statement (excluding nested switch statements)</td>
<td>257</td>
<td>∞</td>
</tr>
<tr>
<td>20</td>
<td>Count of members for one structure or union</td>
<td>127</td>
<td>∞</td>
</tr>
<tr>
<td>21</td>
<td>Count of enumerated type constants in one enumerated type</td>
<td>127</td>
<td>∞</td>
</tr>
<tr>
<td>22</td>
<td>Count of structure or union nesting levels for one structure declaration array</td>
<td>15</td>
<td>∞</td>
</tr>
</tbody>
</table>

The ∞ symbol in the above table indicates the dependence on the memory size available for the system.

*: Although the count of external identifier characters to be identified by the compiler is ∞, only 255 characters are outputted to the assembler. If there are identifiers whose 254 leading characters are the same, an error may occur in the assembler.
CHAPTER 5 EXTENDED LANGUAGE SPECIFICATIONS
It is conceivable that a user program may be executed in an environment where the operating system exists or executed while no operating system support is provided. In an environment in which the operating system exists, it is necessary to prepare the setup process suitable for the environment. This chapter describes the user program execution procedure to be performed in an environment where no operating system exists.

6.1 EXECUTION PROCESS OVERVIEW
6.2 STARTUP ROUTINE CREATION
CHAPTER 6 EXECUTION ENVIRONMENT

6.1 EXECUTION PROCESS OVERVIEW

In an environment where no operating system exists, it is necessary to prepare the startup routine which initiates user program execution.

Execution Process Overview

The main functions to be incorporated into the startup routine are as follows:

- Environment Initialization Necessary for Program Operation
  This initialization must be described by the assembler and completed before user program execution.

- User Program Calling
  The void main(void), which is normally used as the function that the startup routine calls in the program start process, is to be called.

- Shutdown Process
  After a return from the user program is made, the shutdown process necessary for the system is to be performed to accomplish program termination.

Figure 6.1-1 shows the relationship between the startup routine and user function calling.

![Figure 6.1-1 Relationship between Startup Routine and User Function Calling](image)

The precautions to be observed in startup routine preparation are described below.

- Stack
  When the user program is executed, the stack is used for return address, argument storage area, automatic variable area, and register saving, etc. The stack must therefore be provided with an adequate space.

- Register
  When the startup routine calls the user program, it is essential that stack pointer setup be completed. The user program operates on the presumption that the stack top is set as the stack pointer. Further, when the startup routine returns from the user program, the register status is as shown in Table 6.1-1. This is because the employed interface is the same as for register guarantee at the time of function calling.
For register guarantee, see 4.7.5, “Register (stack argument passing)” and 4.8.5, “Register (register argument passing)”. If the guarantee of a register is called for by the system while the value of that register is not guaranteed by the user program, it is necessary to guarantee the value by the startup routine to initiate calling.

Table 6.1-1  Register Status Prevailing at Return from User Program

<table>
<thead>
<tr>
<th>Register</th>
<th>Value Guarantee at Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Not provided</td>
</tr>
<tr>
<td>RW0 to RW2</td>
<td>Provided*</td>
</tr>
<tr>
<td>RW3</td>
<td>Provided</td>
</tr>
<tr>
<td>RW4 and RW5</td>
<td>Not provided</td>
</tr>
<tr>
<td>RW6 and RW7</td>
<td>Provided</td>
</tr>
<tr>
<td>USP (SSP)</td>
<td>Provided</td>
</tr>
</tbody>
</table>

*: When the -rp option is specified, the guarantee of the value of RW0 and RW1 is “Not provided”.
6.2 STARTUP ROUTINE CREATION

This section describes the processes necessary for startup routine creation.

■ Startup Routine Creation

1. Register Initial Setup
   Perform initial setup for RP, ILM, DPR, SSB, SSP, DTB, USB, and USP. The register bank uses one or more. Please set DTB to 0.

2. Data Area Initialization
   The C language specification guarantees the initialization of external variables without the initial value and static variables to 0. Therefore, initialize the data area to 0.

   For the initialization of __far type qualified variable sections, the compiler generates the DCLEAR sections. These sections sequentially store the start addresses of the sections to be cleared to zero and the section sizes. Therefore, use this section when initialization to zero is intended.

   For zero-clearing a section using the DCLEAR section, see Figure 6.2-1. The DATA and DIRDATA sections cannot be zero-cleared by this method, so they should be zero-cleared by another method.

3. Initialization Data Area Duplication
   When incorporating constant data or program into ROM, the default data positioned in the ROM area needs to be copied to the variable area (RAM area).

   For the initialization of __far type qualified, initial value attached variable sections, the compiler generates the DTRANS section. This section sequentially stores the initial value storage section start address, copy destination section start address, and section size data. Therefore, use this section when performing the initial value duplication process.

   For initialization of a section using the DTRANS section, see Figure 6.2-2. The INIT and DIRINIT sections cannot be initialized by this method, so they should be initialized by another method.

4. Library Initial Setup
   When using the libraries, open a file for standard input/output. For details, see 8.2, "INITIALIZATION/TERMINATION PROCESS REQUIRED FOR LIBRARY USE".

5. User Program Calling
   Call the user program.

6. Program Shutdown Process
   The close process must be performed for opened files. The normal end and abnormal end processes must be prepared in accordance with the system.
< Caution >

When the variable area to which _far was type qualified is initialized by using DCLEAR and the DTRANS section that the compiler generated, it is necessary to note it in order of the link of the startup routine.

Please link the startup routine from which module in which the variable to which _far is type qualified is defined ahead.
CHAPTER 7   LIBRARY OVERVIEW

This chapter outlines the C libraries by describing the organization of files provided by the libraries and the relationship to the system into which the libraries are incorporated.

7.1 FILE ORGANIZATION
7.2 RELATIONSHIP TO LIBRARY INCORPORATING SYSTEM
CHAPTER 7 LIBRARY OVERVIEW

7.1 FILE ORGANIZATION

This section describes the files furnished by the libraries.

■ File Organization

The following types of library files and header files are provided.

- Library Files
  Table 7.1-1, Table 7.1-2 list the general-purpose standard library file. Table 7.1-3, Table 7.1-4 list the simulator debugger low-level function library file.

**Table 7.1-1 General-purpose Standard Library file (for stack argument passing)**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Memory Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib907s.lib lib905s.lib lib904s.lib lib902s.lib</td>
<td>For small model</td>
</tr>
<tr>
<td>lib907m.lib lib905m.lib lib904m.lib lib902m.lib</td>
<td>For medium model</td>
</tr>
<tr>
<td>lib907c.lib lib905c.lib lib904c.lib lib902c.lib</td>
<td>For compact model</td>
</tr>
<tr>
<td>lib907l.lib lib905l.lib lib904l.lib lib902l.lib</td>
<td>For large model</td>
</tr>
<tr>
<td>lib907sr.lib lib905sr.lib lib904sr.lib lib902sr.lib</td>
<td>For small model ramconst*</td>
</tr>
<tr>
<td>lib907mr.lib lib905mr.lib lib904mr.lib lib902mr.lib</td>
<td>For medium model ramconst*</td>
</tr>
</tbody>
</table>

**Table 7.1-2 General-purpose Standard Library file (for register argument passing)**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Memory Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib907s_rp.lib lib905s_rp.lib lib904s_rp.lib lib902s_rp.lib</td>
<td>For small model</td>
</tr>
<tr>
<td>lib907m_rp.lib lib905m_rp.lib lib904m_rp.lib lib902m_rp.lib</td>
<td>For medium model</td>
</tr>
<tr>
<td>lib907c_rp.lib lib905c_rp.lib lib904c_rp.lib lib902c_rp.lib</td>
<td>For compact model</td>
</tr>
<tr>
<td>lib907l_rp.lib lib905l_rp.lib lib904l_rp.lib lib902l_rp.lib</td>
<td>For large model</td>
</tr>
<tr>
<td>lib907sr_rp.lib lib905sr_rp.lib lib904sr_rp.lib lib902sr_rp.lib</td>
<td>For small model ramconst*</td>
</tr>
<tr>
<td>lib907mr_rp.lib lib905mr_rp.lib lib904mr_rp.lib lib902mr_rp.lib</td>
<td>For medium model ramconst*</td>
</tr>
</tbody>
</table>

**Table 7.1-3 Simulator Debugger Low-level Function Library File (for stack argument passing)**

<table>
<thead>
<tr>
<th>File Name</th>
<th>Memory Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib907sif.lib lib905sif.lib lib904sif.lib lib902sif.lib</td>
<td>For small model</td>
</tr>
<tr>
<td>lib907mif.lib lib905mif.lib lib904mif.lib lib902mif.lib</td>
<td>For medium model</td>
</tr>
<tr>
<td>lib907cif.lib lib905cif.lib lib904cif.lib lib902cif.lib</td>
<td>For compact model</td>
</tr>
</tbody>
</table>
Table 7.1-3  Simulator Debugger Low-level Function Library File (for stack argument passing)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Memory Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib907sif_rp.lib lib905sif_rp.lib lib904sif_rp.lib lib902sif_rp.lib</td>
<td>For small model</td>
</tr>
<tr>
<td>lib907mif_rp.lib lib905mif_rp.lib lib904mif_rp.lib lib902mif_rp.lib</td>
<td>For medium model ramconst*</td>
</tr>
<tr>
<td>lib907cif_rp.lib lib905cif_rp.lib lib904cif_rp.lib lib902cif_rp.lib</td>
<td>For compact model ramconst*</td>
</tr>
<tr>
<td>lib907lif_rp.lib lib905lif_rp.lib lib904lif_rp.lib lib902lif_rp.lib</td>
<td>For large model ramconst*</td>
</tr>
</tbody>
</table>

Table 7.1-4  Simulator Debugger Low-level Function Library File (for register argument passing)

<table>
<thead>
<tr>
<th>File Name</th>
<th>Memory Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib907sif_rp.lib lib905sif_rp.lib lib904sif_rp.lib lib902sif_rp.lib</td>
<td>For small model</td>
</tr>
<tr>
<td>lib907mif_rp.lib lib905mif_rp.lib lib904mif_rp.lib lib902mif_rp.lib</td>
<td>For medium model</td>
</tr>
<tr>
<td>lib907cif_rp.lib lib905cif_rp.lib lib904cif_rp.lib lib902cif_rp.lib</td>
<td>For compact model</td>
</tr>
<tr>
<td>lib907lif_rp.lib lib905lif_rp.lib lib904lif_rp.lib lib902lif_rp.lib</td>
<td>For large model</td>
</tr>
<tr>
<td>lib907srif_rp.lib lib905srif_rp.lib lib904srif_rp.lib lib902srif_rp.lib</td>
<td>For small model</td>
</tr>
<tr>
<td>lib907mrif_rp.lib lib905mrif_rp.lib lib904mrif_rp.lib lib902mrif_rp.lib</td>
<td>For medium model</td>
</tr>
</tbody>
</table>

*: The ramconst libraries serve programs for which the -ramconst option is specified. For the details of -ramconst, see 3.5.6, “Output Object Related Options”.
CHAPTER 7 LIBRARY OVERVIEW

- Header Files

The header files are shown as follows;

assert.h
ctype.h
erro.h
float.h
limits.h
math.h
setjmp.h
stdarg.h
stddef.h
stdio.h
stdlib.h
string.h

The following three header files define the macros and types that are used when the standard library calls the low-level function library.

fcntl.h
unistd.h
sys/types.h

■ Library Section Names

The library section names vary with the memory model. Table 7.1-5 shows the section names used by the libraries.

Table 7.1-5  Library Section Name

<table>
<thead>
<tr>
<th>Section Type</th>
<th>Small</th>
<th>Medium</th>
<th>Compact</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code section</td>
<td>CODE</td>
<td>LIBCODE</td>
<td>CODE</td>
<td>LIBCODE</td>
</tr>
<tr>
<td>Data section</td>
<td>DATA</td>
<td>DATA</td>
<td>LIBDATA</td>
<td>LIBDATA</td>
</tr>
<tr>
<td>Initial value of DINIT</td>
<td>DCONST</td>
<td>DCONST</td>
<td>LIBDCONST</td>
<td>LIBDCONST</td>
</tr>
<tr>
<td>Initialized section</td>
<td>INIT</td>
<td>INIT</td>
<td>LIBINIT</td>
<td>LIBINIT</td>
</tr>
<tr>
<td>Constant section</td>
<td>CONST</td>
<td>CONST</td>
<td>LIBCONST</td>
<td>LIBCONST</td>
</tr>
<tr>
<td>RAM area of CCONST</td>
<td>CINIT</td>
<td>CINIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7.2 RELATIONSHIP TO LIBRARY INCORPORATING SYSTEM

This section describes the relationship between the libraries and library incorporating system.

■ System-dependent Processes

File input/output, memory management, and program termination procedures are the processes dependent on the system. When such system-dependent processes are needed, the libraries issue a call as a low-level function. For the details of low-level functions, see CHAPTER 8, "LIBRARY INCORPORATION".

When using the libraries, prepare such low-level functions in accordance with the system.

■ Low-level Function (System-dependent Process) Types

The low-level function types and their roles are summarized below. For the detailed feature descriptions of low-level functions, see 8.5, "LOW-LEVEL FUNCTION SPECIFICATIONS".

- **open**: Function for opening a file in the system
- **close**: Function for closing a file in the system
- **read**: Function for reading characters from a file
- **write**: Function for writing characters into a file
- **lseek**: Function for changing the file position
- **isatty**: Function for checking whether a file is a terminal file
- **sbrk**: Function for dynamically acquiring/changing the memory
- **_exit**: Function for normal program ending
- **_abort**: Function for abnormal program ending
CHAPTER 8 LIBRARY INCORPORATION

This chapter describes the processes and functions to be prepared for library use.

8.1 LIBRARY INCORPORATION OVERVIEW
8.2 INITIALIZATION/TERMINATION PROCESS REQUIRED FOR LIBRARY USE
8.3 LOW-LEVEL FUNCTION TYPES
8.4 STANDARD LIBRARY FUNCTIONS AND REQUIRED PROCESS/LOW-LEVEL FUNCTIONS
8.5 LOW-LEVEL FUNCTION SPECIFICATIONS
CHAPTER 8 LIBRARY INCORPORATION

8.1 LIBRARY INCORPORATION OVERVIEW

This section outlines library incorporation.

- Processes and Functions Required for Library Use

  File input/output, memory management, and program termination procedures are the processes dependent on the system. Therefore, when such system-dependent processes are needed, such processes are separated from standard library by library, and whenever such processes are needed, they will be called as a low-level function. Further, the stream area initialization and other processes are required for library use.

  The following processes and functions must be prepared for library use.

  • Stream area initialization
  • Standard input/output and standard error output file open and close processes
  • Low-level function creation

  At the time of library incorporation, the above processes and functions must be prepared in accordance with the system.
This section describes the initialization/termination process required for library use.

■ Initialization/Termination Process

Some standard library functions require the following processes, which are detailed in this section.

- Stream area initialization
- Standard input/output and standard error output file opening and closing

For required functions, see 8.4, “STANDARD LIBRARY FUNCTIONS AND REQUIRED PROCESS/LOW-LEVEL FUNCTIONS”.

■ Stream Area Initialization

The _stream_init function initializes the stream area. This function must be called by the startup routine to initialize the stream area.

```
void _stream_init( void);
```

■ Standard Input/Output and Standard Error Output File Opening and Closing

The standard input/output and standard error output are to be opened and closed in a program. Therefore, the opening process must be performed before main function calling, and the closing process must be performed after main function execution.

Use the startup routine to perform the opening process before main function calling and the closing process after main function execution.

However, the _stream_init function correlates the file numbers 0, 1, and 2 to the stdin, stdout, and stderr streams. Therefore, the opening process need not be performed when the system’s standard input, standard output, and standard error output are opened as the file numbers 0, 1, and 2.

If the system’s standard input/output and standard error output are not opened or the file numbers do not match, perform the following process to open the system’s files.

```
freopen( "Standard input name" , "r", stdin );
freopen( "Standard output name" , "w", stdout);
freopen( "Standard error output name", "w" stderr);
```

Error detection concerning the above process should be conducted as needed.

Further, the file names specified by the open function must be written as the standard input/output and standard error output names.

For the closing process, use the fclose function.
8.3 LOW-LEVEL FUNCTION TYPES

This section outlines the standard library functions and required low-level functions. The following six types of low-level functions are required for the standard library functions.

- **File opening and closing** (open and close)
- **Input and output relative to file** (read and write)
- **File position change** (lseek)
- **File inspection** (isatty)
- **Memory area dynamic acquisition** (sbrk)
- **Program abnormal end and normal end** (_abort and _exit)

The above processes are called from the associated standard libraries to manipulate the system’s actual files or exercise program execution control.

---

## Low-level Function Types

- **File Opening and Closing**
  
  When the open function is called, the fopen and all other file opening functions open the system’s actual files.

  In like manner, the fclose and all other file closing functions close the system’s actual files when the close function is called.

- **Input and Output Relative to File**
  
  The scanf, printf, and other input/output functions perform input/output operations relative to the system’s actual files when the read and write functions are called.

- **File Position Change**
  
  The fseek and other file position manipulation functions acquire or change the system’s actual file positions when the lseek function is called.

- **File Inspection**
  
  Checks whether an open file is a terminal file.

- **Memory Area Dynamic Acquisition**
  
  The malloc and other memory area dynamic acquisition functions acquire or free specific memory areas when the sbrk function is called.

- **Program Abnormal End and Normal End**
  
  The abort function and exit function call the _abort function and _exit function, respectively, as the termination process.
8.4 STANDARD LIBRARY FUNCTIONS AND REQUIRED PROCESS/LOW-LEVEL FUNCTIONS

This section describes the standard library functions and associated initialization/termination processes and low-level functions.

Table 8.4-1 Standard Library Functions and Required Processes/Low-level Functions

<table>
<thead>
<tr>
<th>Standard Library Function</th>
<th>Low-level Function</th>
<th>Initialization/Termination Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert ()</td>
<td>open ()</td>
<td>close ()</td>
</tr>
<tr>
<td>abort ()*</td>
<td>read ()</td>
<td>write ()</td>
</tr>
<tr>
<td></td>
<td>lseek ()</td>
<td>isatty ()</td>
</tr>
<tr>
<td></td>
<td>sbrk ()</td>
<td>_abort ()</td>
</tr>
</tbody>
</table>

| All stdio.h functions     | open ()           | close ()                          |
|                           | read ()           | write ()                          |
|                           | lseek ()          | isatty ()                         |

| calloc ()                 | sbrk ()           | Stream area initialization process standard input/output and standard error output opening and closing |

| malloc ()                 | sbrk ()           | Stream area initialization process standard input/output and standard error output opening and closing |

| realloc ()                | sbrk ()           | Stream area initialization process standard input/output and standard error output opening and closing |

| free ()                   | sbrk ()           | Stream area initialization process standard input/output and standard error output opening and closing |

| exit ()*                  | open ()           | close ()                          |
|                           | read ()           | write ()                          |
|                           | lseek ()          | isatty ()                         |
|                           | sbrk ()           | _exit ()                          |

*: When the abort function and exit function are called, they perform the closing process for open files. Therefore, the file manipulation related low-level functions (open, close, read, write, lseek, and sbrk) and stream area initialization and like processes are required.

In a program that is not using a file, the _abort function can be directly called instead of the abort function.

In a program for which function registration is not completed using the atexit function, the _exit function can be directly called instead of the exit function while no file is being used.

In the above instances, file manipulation related low-level function use and stream area initialization are not required.
8.5 LOW-LEVEL FUNCTION SPECIFICATIONS

There are various low-level functions. The open, close, read, write, lseek, and isatty functions provide file processing. The sbrk function provides memory area dynamic allocation. The _exit or _abort function is used to terminate a program by calling the exit or abort function. These low-level functions must be created to suit the system.

− Low-level Function Specifications
  Create the low-level functions in compliance with the specifications stated in this section.
8.5 LOW-LEVEL FUNCTION SPECIFICATIONS

8.5.1 open Function

Create the open function in compliance with the specifications stated in this section.

```c
#include <fcntl.h>
int open( char *fname, int fmode, int p );
```

- **open Function**
  - **[Explanation]**
    - In the mode specified by fmode, open the file having the name specified by fname. For fmode specifying, a combination of the following flags (logical OR) is used. The third argument `p` is a permission mode specified for the file when the specified file is newly made. Whenever standard function fopen and freopen call the open function, 0777 is passed.
      - `O_RDONLY`:
        - Opens a read-only file
      - `O_WRONLY`:
        - Opens a write-only file
      - `O_RDWR`:
        - Opens a read/write file
    - The above three flags are to be exclusively specified.
      - `O_CREAT`:
        - Create this flag when the specified file does not exist. If the specified file already exists, ignore this flag.
      - `O_TRUNC`:
        - If any data remains in the file, discard such data to empty the file.
      - `O_APPEND`:
        - Selects the append mode for file opening
        - The file position prevailing at the time of opening must be set so as to indicate the end of the file. When writing into a file placed in this mode, start writing at the end of the file without regard to the current file position.
      - `O_BINARY`:
        - Specifies a binary file
        - Therefore, the file opened must be treated as a binary file. Files for which this is not specified must be treated as text files.
    - When the name for standard input/output and standard error output, which is determined for system environment setup, is specified as the file name for the first parameter, allocate the standard input/output and standard error output to the file to be opened.
  - **[Return Value]**
    - When file opening is successfully done, the file number must be returned. If file opening is not successfully done, on the other hand, the value -1 must be returned.
8.5.2 close Function

Create the close function in compliance with the specifications stated in this section.

```c
int close( int fileno);
```

- **close Function**
  - **[Explanation]**
    The closing process must be performed for the file specified by `fileno`.
  - **[Return Value]**
    When file closing is successfully done, the value 0 must be returned. If file closing is not successfully done, the value -1 must be returned.
8.5.3 read Function

Create the `read` function in compliance with the specifications stated in this section.

```c
int read(int fileno, char *buf, int size);
```

- **read Function**
  - **[Explanation]**
    - From the file specified by `fileno`, size-byte data must be input into the area specified by `buf`.
    - If the text file new line character is other than `\n` in the system environment at this time, perform setup with the new line character converted to `\n` by the `read` function.
  - **[Return Value]**
    - When the input from the file is successfully done, the input character count must be returned.
    - If the input from the file is not successfully done, the value -1 must be returned. If the file ends in the middle of the input sequence, a value smaller than `size` can be returned as the input character count.
8.5.4 write Function

Create the write function in compliance with the specifications stated in this section.

```c
int write (int fileno, char *buf, int size);
```

### write Function

[Explanation]

To the file specified by fileno, size-byte data in the area specified by buf must be outputted. If the file is opened in the append mode, the output must always be appended to the end of the file. If the text file new line character is other than \n in the system environment at this time, the output must be generated with the system environment new line character converted to \n by the write function.

[Return Value]

When the output to the file is successfully done, the output character count must be returned. If it is not successfully done, the value -1 must be returned.
8.5.5 Lseek Function

Create the lseek function in compliance with the specifications stated in this section.

```c
#include <unistd.h>
long int lseek( int fileno, off_t offset, int whence);
```

[**Iseek Function**]

[Explanation]

The file specified by fileno must be moved to a position that is offset bytes away from the position specified by whence. The file position is determined according to the byte count from the beginning of the file. The following three positions are to be specified by whence.

- SEEK_CUR:
  Adds the offset value to the current file position
- SEEK_END:
  Adds the offset value to the end of the file
- SEEK_SET:
  Adds the offset value to the beginning of the file

[Return Value]

When the file position is successfully changed, the new file position must be returned. If it is not successfully changed, -1L must be returned.
8.5.6 isatty Function

Create the isatty function in compliance with the specifications stated in this section.

```c
int isatty(int fileno);
```

- **isatty Function**

  [Explanation]
  
  The file specified by fileno is to be checked to see whether it is a terminal file. When the file is a terminal file, true must be returned. If not, false must be returned.

  [Return Value]
  
  When the specified file is a terminal file, true must be returned. If not, false must be returned.
8.5.7 sbrk Function

Create the sbrk function in compliance with the specifications stated in this section.
char *sbrk(int size);

- sbrk Function

  [Explanation]
  The existing area must be enlarged by size bytes. If size is a negative quantity, the area must be reduced.
  If the sbrk function has not been called, furnish a size-byte area.
  The area varies as shown below with sbrk function calling.

  ![Figure 8.5-1 Area Change Brought About by sbrk Function Calling]

  Before change | After a change effected by a plus size value | After a change effected by a minus size value

  | Low
  | Existing area
  | *1→
  | High

  Return value = *1 (the end address of the area prevailing before the area change) + 1

  [Return Value]
  When the area change is successfully made, the value to be returned must be determined by adding the value 1 to the end address of the area prevailing before the area change. If the sbrk function has not been called, the start address of the acquired area must be returned. If the area change is not successfully made, the value (char)-1 must be returned.
8.5.8 _exit Function

Create the _exit function in compliance with the specifications stated in this section.

```c
#include <stdlib.h>
void _exit( int status);
```

**_exit Function**

[Explanation]

The _exit function must bring the program to a normal end. When the status value is 0 or in the case of EXIT_SUCCESS, the successful end state must be returned to the system environment. In the case of EXIT_FAILURE, the unsuccessful end state must be returned to the system environment.

[Return Value]

The _exit function does not return to the caller.
8.5.9 _abort Function

Create the _abort function in compliance with the specifications stated in this section.

void _abort( void);

 ■ _abort Function

 [Explanation]

 The _abort function must bring the program to an abnormal end.

 [Return Value]

 The _abort function does not return to the caller.
CHAPTER 9 COMPILER-DEPENDENT SPECIFICATIONS

This chapter describes the specifications that vary with the compiler. The descriptions set forth in this chapter relate to the JIS requirements which are standardized on the basis of the ANSI standard.

9.1 COMPILER-DEPENDENT LANGUAGE SPECIFICATION DIFFERENTIALS
9.2 FLOATING-POINT DATA FORMAT AND EXPRESSIBLE VALUE RANGE
9.3 FLOATING-POINT OPERATION DUE TO THE RUNTIME LIBRARY FUNCTION
### 9.1 COMPILER-DEPENDENT LANGUAGE SPECIFICATION DIFFERENTIALS

Table 9.1-1 lists the compiler-dependent language specification differentials.

**Compiler-dependent Language Specification Differentials**

<table>
<thead>
<tr>
<th>Specification Differentials</th>
<th>Associated JIS Requirements</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Japanese language process support and code system</td>
<td>5.2.1 Character Set 6.1.2 Identifier</td>
<td>No support EUC and SJIS entries can be made only in the comment.</td>
</tr>
<tr>
<td>Recognized character count of an identifier with an external binding</td>
<td>6.1.2 Identifier</td>
<td>254 leading characters</td>
</tr>
<tr>
<td>Differentiation between upper- and lower-case alphabetical characters of an identifier with an external binding</td>
<td>6.1.2 Identifier</td>
<td>Treated as different characters</td>
</tr>
<tr>
<td>Character set element expression code system</td>
<td>6.1.3 Constant</td>
<td>ASCII code</td>
</tr>
<tr>
<td>Char type treatment and expressible value range</td>
<td>6.2.1.1 Character Type and Integer Type</td>
<td>Unsigned*¹ 0 to 255</td>
</tr>
<tr>
<td>Floating-point data formats and sizes</td>
<td>6.1.2.5 Type</td>
<td>IEEE type*² 4 bytes 8 bytes</td>
</tr>
<tr>
<td>Whether or not to treat the start bit as signed bit when following types specified as bit field char, short int, int, and long int type</td>
<td>6.5.2.1 Structure Specifier and UnionSpecifier</td>
<td>Not treated as a sign*²</td>
</tr>
<tr>
<td>Types that can be specified as bit field</td>
<td>6.5.2.1 Structure Specifier and UnionSpecifier</td>
<td>char type signed char type unsigned char type short int type unsigned short int type int type unsigned int type long int type unsigned long int type</td>
</tr>
<tr>
<td>Structure or union type member boundary alignment value</td>
<td>6.5.2.1 Structure Specifier and UnionSpecifier</td>
<td>1 byte 1 byte 1 byte 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes 2 bytes</td>
</tr>
</tbody>
</table>
### Table 9.1-1  Compiler-dependent Language Specification Differentials (Continued)

<table>
<thead>
<tr>
<th>Specification Differentials</th>
<th>Associated JIS Requirements</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character constant expression code system for pre-processor</td>
<td>6.8.1 Conditional Acquisition</td>
<td>ASCII code</td>
</tr>
<tr>
<td>Registers that can be specified within asm statement</td>
<td></td>
<td>A, AL, and AH*3</td>
</tr>
<tr>
<td>ANSI-compliant standard library function support</td>
<td></td>
<td>Refer to the volume entitled Libraries</td>
</tr>
</tbody>
</table>

*1: Alterable through option use.

*2: See 9.2, “FLOATING-POINT DATA FORMAT AND EXPRESSIBLE VALUE RANGE”.

*3: The other registers can be used when they are saved and recovered by the user.
9.2 FLOATING-POINT DATA FORMAT AND EXPRESSIBLE VALUE RANGE

Table 9.2-1 shows the floating-point data format and expressible value range.

### Floating-point Data Format and Expressible Value Range

<table>
<thead>
<tr>
<th>Floating-point Data Format</th>
<th>Expressible Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>float type</td>
<td>The exponent part is a value between $2^{-126}$ and $2^{+127}$. The fractional part of the mantissa (the integer portion is normalized to 1) is binary and has 24-digit accuracy.</td>
</tr>
<tr>
<td>double type</td>
<td>The exponent part is a value between $2^{-1022}$ and $2^{+1023}$. The fractional part of the mantissa (the integer part is normalized to 1) is binary and has 53-digit accuracy.</td>
</tr>
<tr>
<td>long double type</td>
<td>The exponent part is value between $2^{-1022}$ and $2^{+1023}$. The fractional part of the mantissa (the integer part is normalized to 1) is binary and has 53-digit accuracy.</td>
</tr>
</tbody>
</table>
9.3 FLOATING-POINT OPERATION DUE TO THE RUNTIME LIBRARY FUNCTION

All floating-point operations, except for ones calculated in the compilation time, are done by the runtime library functions. Although those functions are designed referring to ANSI/IEEE Std754-1985, they do not completely conform to it. This section describes the differences between the specification of the floating-point runtime library functions and ANSI/IEEE Std754-1985.

■ Arithmetic operation (addition, subtraction, multiplication, and division)
  ○ Rounding of the resultant mantissa part
    Round-to-nearest mode, only.
  ○ Denormalized number
    If the left operand is a denormalized number, it is assumed to be zero with the same sign. If the right operand is a denormalized number, it is assumed to be zero with the same sign, too. In some cases, the denormalized number with the correct sign is returned rather than the strict zero.
  ○ Resultant value under the underflow exception
    It is assumed that the underflow exception occurs when the absolute value of true operation result is too small to be represented as the normalized number. In that case, zero with the correct sign is returned.
  ○ Resultant value under the overflow exception
    Infinity with the correct sign is returned.
  ○ Resultant value under the invalid operation exception
    NaN (Not a number) is returned. In the floating-point runtime library, any routines do not distinguish SNaN (Signaling NaN) and QNaN (Quiet NaN).
  ○ Interrupt at operation exception
    No interrupt occur.
  ○ Status flag
    Not supported.

■ Comparison
  ○ Denormalized number
    The denormalized number is treated as zero with the same sign.
  ○ Comparison result under the invalid operation exception
    The library function returns uncertain result.
CHAPTER 9 COMPILER-DEPENDENT SPECIFICATIONS

 Interrupt at operation exception
  No interrupt occur.

 Status flag
  Not supported.

■ Type conversion (integer -> floating-point number)

 Rounding of the resultant mantissa part
  Round-to-nearest mode, only.

 Interrupt at operation exception
  No interrupt occur.

 Status flag
  Not supported.

■ Type conversion (floating-point number -> integer)

 Resultant value under the invalid operation exception
  Uncertain value is returned.

 Interrupt at operation exception
  No interrupt occur.

 Status flag
  Not supported.

■ Type conversion (floating-point number -> floating-point number)

 Rounding of the resultant mantissa part
  Round-to-nearest mode, only.

 Denormalized number
  If the converting value is a denormalized number, it is treated as zero with the same sign. In some cases, the denormalized number is returned rather than the strict zero.

 Resultant value under the underflow exception
  It is assumed that the underflow exception occurs when the absolute value of true operation result is too small to be represented as the normalized number. In that case, zero with the correct sign is returned.

 Resultant value under the overflow exception
  Infinity with the correct sign is returned.

 Resultant value under the invalid operation exception
  NaN (Not a Number) is returned. In the floating-point runtime library, any routines do not distinguish SNaN (Signaling NaN) and QNaN (Quiet NaN).
9.3 FLOATING-POINT OPERATION DUE TO THE RUNTIME LIBRARY FUNCTION

- **Interrupt at operation exception**
  
  No interrupt occur.

- **Status flag**
  
  Not supported.
CHAPTER 10 SIMULATOR DEBUGGER LOW-LEVEL FUNCTION LIBRARY

The simulator debugger low-level function library is a library of the low-level functions which are necessary when the standard library is used with the simulator debugger. This chapter describes how to use the simulator debugger low-level function library.

10.1 LOW-LEVEL FUNCTION LIBRARY OVERVIEW
10.2 LOW-LEVEL FUNCTION LIBRARY USE
10.3 LOW-LEVEL FUNC. FUNCTION
10.4 LOW-LEVEL FUNCTION LIBRARY CHANGE
10.1 LOW-LEVEL FUNCTION LIBRARY OVERVIEW

This section outlines the low-level function library.

■ Low-level Function Library Overview

The low-level function library offers the functions that are necessary when the standard library is used with the simulator debugger. The main functions are as follows.

- File manipulation functions based on I/O port simulation (open, close, read, write, lseek, and isatty)
- Dynamic memory allocation function (sbrk)

In the simulator debugger, the program executed cannot terminate its own execution. Therefore, prepare the _abort and _exit functions.

■ File System Overview

The low-level function library uses the I/O port simulation function of the simulator debugger to carry out standard input/output operations and input/output operations relative to files. These operations are completed by performing input/output operations relative to one I/O port area which is regarded as one file.

When the open function is called, it allocates a 1-byte area of the I/O port simulation area (I/O section) defined by the low-level function library and returns as the file number the offset from the beginning of the allocated area.

The read function and write function perform input/output operations relative to the 1-byte area allocated by the open function.

Input/output operations can be performed relative to the standard input/output and files when such standard input/output and files are allocated to the above-mentioned area prior to program execution using simulator debugger commands (set inport and set outport).

The close function frees an already allocated area to render it reusable.

Since the file position cannot be changed in the simulator debugger, the value -1 is always returned for the lseek function.

■ Area Management

An already acquired external variable area is used as the area returned by the sbrk function. When the sbrk function is called, area allocation begins with the lowest address of the area.
10.2 LOW-LEVEL FUNCTION LIBRARY USE

This section describes the load module creation and simulator debugger setup procedures to be performed for low-level function library use.

■ Initialization
No initialization is required except for _stream_init function calling.
When creating the startup routine in accordance with the system, call the _stream_init function prior to main function calling.

■ Load Module Creation
After completing creation of the necessary program, compile and link all the necessary modules. Link the following libraries in accordance with the memory model. Select a low-level library in accordance with the host that starts the simulator debugger.

Table 10.2-1 Libraries to be Linked for Load Module Creation

<table>
<thead>
<tr>
<th>Memory Model</th>
<th>-rp</th>
<th>ramconst</th>
<th>Standard Library</th>
<th>Low-level Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small model</td>
<td></td>
<td>Not specified</td>
<td>lib907s.lib</td>
<td>lib907sif.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib905s.lib</td>
<td>lib905sif.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib904s.lib</td>
<td>lib904sif.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib902s.lib</td>
<td>lib902sif.lib</td>
</tr>
<tr>
<td></td>
<td>Specified</td>
<td>lib907sr.lib</td>
<td>lib907srif.lib</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specified</td>
<td>lib905sr.lib</td>
<td>lib905srif.lib</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specified</td>
<td>lib904sr.lib</td>
<td>lib904srif.lib</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specified</td>
<td>lib902sr.lib</td>
<td>lib902srif.lib</td>
<td></td>
</tr>
<tr>
<td>Specified</td>
<td></td>
<td>Not specified</td>
<td>lib907s_rp.lib</td>
<td>lib907sif_rp.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib905s_rp.lib</td>
<td>lib905sif_rp.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib904s_rp.lib</td>
<td>lib904sif_rp.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib902s_rp.lib</td>
<td>lib902sif_rp.lib</td>
</tr>
<tr>
<td>Specified</td>
<td></td>
<td>Not specified</td>
<td>lib907sr_rp.lib</td>
<td>lib907sif_rp.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib905sr_rp.lib</td>
<td>lib905sif_rp.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib904sr_rp.lib</td>
<td>lib904sif_rp.lib</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lib902sr_rp.lib</td>
<td>lib902sif_rp.lib</td>
</tr>
</tbody>
</table>
### Table 10.2-1 Libraries to be Linked for Load Module Creation

<table>
<thead>
<tr>
<th>Memory Model</th>
<th>-rp</th>
<th>ramconst</th>
<th>Standard Library</th>
<th>Low-level Library</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Medium model</strong></td>
<td>Not specified</td>
<td>lib907m.lib lib905m.lib lib904m.lib lib902m.lib</td>
<td>lib907mif.lib lib905mif.lib lib904mif.lib lib902mif.lib</td>
<td></td>
</tr>
<tr>
<td>Specified</td>
<td>Specified</td>
<td>lib907mif.lib lib905mif.lib lib904mif.lib lib902mif.lib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specified</td>
<td>Specified</td>
<td>lib907mif_rp.lib lib905mif_rp.lib lib904mif_rp.lib lib902mif_rp.lib</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Compact model</strong></td>
<td>Not specified</td>
<td>Not specified</td>
<td>lib907c.lib lib905c.lib lib904c.lib lib902c.lib</td>
<td>lib907cif.lib lib905cif.lib lib904cif.lib lib902cif.lib</td>
</tr>
<tr>
<td>Specified</td>
<td>Not specified</td>
<td>lib907cif_rp.lib lib905cif_rp.lib lib904cif_rp.lib lib902cif_rp.lib</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Large model</strong></td>
<td>Not specified</td>
<td>Not specified</td>
<td>lib907l.lib lib905l.lib lib904l.lib lib902l.lib</td>
<td>lib907lif.lib lib905lif.lib lib904lif.lib lib902lif.lib</td>
</tr>
<tr>
<td>Specified</td>
<td>Not specified</td>
<td>lib907lif_rp.lib lib905lif_rp.lib lib904lif_rp.lib lib902lif_rp.lib</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10.2 LOW-LEVEL FUNCTION LIBRARY USE

Simulator Debugger Setup

Setup for standard input/output use is as follows.

[Example of Debugger Setup]

```
set inport/ascii  0, 0xff,$TERMINAL
set outport/ascii 1, 0xff,$TERMINAL
```

Since the first three areas of the I/O section are used for standard input, standard output, and standard error output, the other files are allocated to the fourth and subsequent areas (the offset from the beginning of the I/O section is 3).

In other words, allocation is performed sequentially in the order of file opening (offset 3, offset 4, etc.). Therefore, perform setup accordingly using the set inport and set outport commands.

To open a.doc as the input file and then open b.doc as the output file, setup as shown below.

```
set inport/ascii 3,h'ff,"a.doc"
set outport/ascii 4,h'ff,"b.doc"
```

Example

Create a program that displays the character string "Hello!!" using the small model, and initiate execution with the simulator debugger.

```
main()
{
    printf("Hello!!\n");
}
```

Create a C-source file named test.c as shown above.

Compile using the following command. Setup the corresponding directory for LIBTOOL.

```
fcc907s test.c -model SMALL -cpu MB90F553A
flnk907s LIBTOOL/start905s.obj test.obj -L LIBTOOL
    -I lib905s.lib lib905sif.lib
    -O test.abs -cpu MB90F553A
```

At completion of the preceding step, test.abs is created. Execute the created file with the simulator debugger.

After startup, input following commands. end is a symbol defined within the startup routine. Create the startup routine object as the one with the debug information.

```
> set inport/ascii h'0,h'ff,$TERMINAL
> set outport/ascii h'1,h'ff,$TERMINAL
> go , end
```

Since standard input is not involved in the above example, the set inport command can be omitted.
This section describes the function specific to the simulator debugger low-level functions.

- **Special I/O Port**

  As far as the low-level functions are concerned, the first three bytes of the I/O section are specified to function as the standard input, standard output, and standard error output, respectively. For such bytes, files No. 1, 2, and 3 are allocated. They are initialized to the opened state.

  Table 10.3-1 shows the predefined I/O port.

  **Table 10.3-1 Predefined I/O Port**

<table>
<thead>
<tr>
<th>Address</th>
<th>File Number</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Standard input</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Standard output</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Standard error output</td>
</tr>
</tbody>
</table>

  The input from the standard input (file No. 0) is outputted to the standard output (file No. 1). The input to the standard input (file No. 0) is discontinued if the new line character `\n` is entered. However, when the input is fed from some other port, the input continues until the required number of characters are read.

- **open Function**

  The open function finds an unused I/O port area and then returns as the file number the area’s offset from the beginning of the I/O section. In such an instance, the file name and open mode are not to be specified. Even if files are opened using the same file name, differing file numbers are assigned to them.

  Files No. 0, 1, and 2 are initialized to the opened state. Therefore, the open function begins allocation with file No. 3 unless files No. 0, 1, and 2 are subjected to the close process. Five files or less from the file number 0 to 4 can be opened at the same time.
### read Function

The read function reads data from the I/O port area specified by the address which is determined by adding the specified file number to the I/O section start address.

The input from file No. 0 is treated as a line input. When the new line character \n is entered, the read function terminates even if the required character count is not reached. Further, this input is outputted to the standard output (file No. 1). The input from a file numbered other than 0 is treated as a block input. Reading continues until the required character count is reached.

### write Function

The write function writes data to the I/O port area specified by the address which is determined by adding the specified file number to the I/O section start address. Unlike the input, the operation does not vary with the I/O port area address.

### lseek Function

The file position cannot be specified in the simulator debugger. Therefore, the value -1, which indicates an unsuccessful file position change, is always returned.

### isatty Function

In the case of file No. 0, 1, or 2, true is returned. In the other cases, false is returned.

### close Function

The close function releases the port related to the specified file number.

### sbrk Function

The simulator debugger does not provide a means of dynamic memory allocation. Therefore, the sbrk function acquires a fixed area and uses it.

To change the area or its size, create an alternative function and substitute it for the sbrk function with a librarian. For details, see 10.4, “LOW-LEVEL FUNCTION LIBRARY CHANGE”.

This section describes how to change the dynamic allocation area (heap).

### Dynamic Allocation Area Change

Locate the following line in the sbrk.c source program list. Change the value in this line to the dynamic allocation area size (in bytes).

```
#define HEEP_SIZE    16*1024
```

Use the following commands to compile and update the library. At compiling, specify the section name shown in Table 7.1-1.

**[For Small Model]**

```
> fcc907s -O sbrk.c -model SMALL -cpu MB90F553A
> flib907s -r sbrk.obj lib905sif.lib -cpu MB90F553A
```

**[For Large Model]**

```
> fcc907s -O sbrk.c -model LARGE -s FAR_CODE=LIBCODE
    -s FAR_DCONST=LIBDCONST -s FAR_INIT=LIBINIT -s FAR_DATA=LIBDATA
    -s FAR_CONST=LIBCONST -cpu MB90F553A
> flib907s -r sbrk.obj lib905lif.lib -cpu MB90F553A
```

### sbrk.c Source Program List

The source program required for changing the dynamic area is shown below. The file name must be sbrk.c.

```
#define HEEP_SIZE       16*1024
static long     brk_siz = 0;
static char     _heep[HEEP_SIZE];
#define         _heep_size      HEEP_SIZE
extern char *sbrk(int size)
{
    if (brk_siz + size > _heep_size || brk_siz + size < 0)
        return((char*)-1);
    brk_siz += size;
    return(_heep + brk_siz - size);
}
```
APPENDIX

The appendix gives a list of types, macros, variables, and functions provided by the library and the operations specific to the libraries (A,B). Notes when F^2MC-16LX CPU is used are described (C). The guide to change the function call interface to the register argument passing is described (D). The list of the error message is described (E).

Appendix A LIST OF TYPE, MACRO, VARIABLE, AND FUNCTION
Appendix B OPERATION SPECIFIC TO LIBRARIES
Appendix C NOTES OF SIGNED DIVISION INSTRUCTION OF F^2MC-16LX CPU
Appendix D GUIDE TO CHANGING FUNCTION-CALL INTERFACE
Appendix E ERROR MESSAGE
This section lists the types, macros, variables, and functions provided by the libraries.

- **assert.h**
  - Function
    - `assert`

- **ctype.h**
  - Macros
    - `isalnum` `isalpha` `iscntrl` `isdigit` `isgraph` `islower` `isprint` `ispunct` `isspace` `isupper` `isxdigit` `tolower` `toupper`

- **errno.h**
  - Macros
    - `EDOM` `ERANGE`
  - Variable
    - `errno`

- **float.h**
  - Macros
    - `FLT_RADIX` `FLT_ROUNDS` `FLT_MANT_DIG` `DBL_MANT_DIG` `LDBL_MANT_DIG` `FLT_MIN_EXP` `DBL_MIN_EXP` `LDBL_MIN_EXP` `FLT_MIN_10_EXP` `DBL_MIN_10_EXP` `LDBL_MIN_10_EXP` `FLT_MAX_EXP` `DBL_MAX_EXP` `LDBL_MAX_EXP` `FLT_MAX_10_EXP` `DBL_MAX_10_EXP` `LDBL_MAX_10_EXP` `FLT_MAX` `DBL_MAX` `LDBL_MAX` `FLT_EPSILON` `DBL_EPSILON` `LDBL_EPSILON` `FLT_MIN` `DBL_MIN` `LDBL_MIN`

- **limits.h**
  - Macros
    - `MB_LEN_MAX` `CHAR_BIT` `SCHAR_MIN` `SCHAR_MAX` ` UCHAR_MAX` `CHAR_MIN` `CHAR_MAX` `INT_MIN` `INT_MAX` ` UINT_MAX` `SHRT_MIN` `SHRT_MAX` `USHRT_MAX` `LONG_MIN` `LONG_MAX` `ULONG_MAX`
Appendix A  LIST OF TYPE, MACRO, VARIABLE, AND FUNCTION

■ math.h

- Macros
  HUGE_VAL  EDOM  ERANGE

- Function
  acos  asin  atan  atan2  cos
  sin  tan  cosh  sinh  tanh
  exp  frexp  ldexp  log  log10
  modf  pow  sqrt  ceil  fabs
  floor  fmod

■ stdarg.h

- Type
  va_list

- Macros
  va_start  va_arg  va_end

■ stddef.h

- Type
  ptrdiff_t  size_t

- Macros
  NULL  offsetof

■ stdio.h

- Type
  ptrdiff_t  size_t  FILE  fpos_t

- Macros
  NULL  EOF  SEEK_SET  SEEK_CUR  SEEK_END
  _IONBF  _IOLBF  _IOFBF  BUFSIZ  stdin
  stdout  stderr  putchar  putc  getchar
  getc  offsetof

- Function
  putchar  putc  getchar  getc  fclose
  fflush  fopen  freopen  setbuf  setvbuf
  fprintf  fscanf  printf  scanf  sprintf
  sscanf  vfprintf  vprintf  vsprintf  fgetc
  fgets  fputc  fputs  gets  puts
  ungetc  fread  fwrite  fgetpos  fseek
  fsetpos  ftell  rewind  clearerr  feof
### APPENDIX

- **stdlib.h**
  - **Type**
    - `ptrdiff_t`
    - `size_t`
    - `div_t`
    - `ldiv_t`
  - **Macros**
    - `NULL`
    - `offsetof`
    - `EXIT_FAILURE`
    - `EXIT_SUCCESS`
    - `RAND_MAX`
  - **Function**
    - `atof`
    - `atoi`
    - ` atol`
    - `strtod`
    - `strtol`
    - `stroull`
    - `rand`
    - `srand`
    - `calloc`
    - `malloc`
    - `realloc`
    - ` abort`
    - `atexit`
    - `exit`
    - `bsearch`
    - `qsort`
    - `abs`
    - `div`
    - `labs`

- **string.h**
  - **Type**
    - `ptrdiff_t`
    - `size_t`
  - **Macros**
    - `offsetof`
  - **Function**
    - `memcpy`
    - `memmove`
    - `strcpy`
    - `strncpy`
    - `strcat`
    - `strncpy`
    - `strchr`
    - `strcspn`
    - `strpbrk`
    - `strrchr`
    - `strspn`
    - `strstr`
    - `strtok`
    - `strlen`

- **fcntl.h**
  - **Macros**
    - `O_RDONLY`
    - `O_WRONLY`
    - `O_RDWR`
    - `O_APPEND`
    - `O_CREAT`
    - `O_TRUNC`
    - `O_BINARY`

- **unistd.h**
  - **Macros**
    - `SEEK_SET`
    - `SEEK_CUR`
    - `SEEK_END`

- **setjmp.h**
  - **Type**
    - `jmp_buf`
  - **Macros**
    - `setjmp`
  - **Function**
    - `longjmp`

- **sys/types.hj**
  - **Type**
    - `off_t`
Appendix B OPERATIONS SPECIFIC TO LIBRARIES

This section describes the operations specific to the libraries.

Operations Specific to Libraries

1. Diagnostic information printed out by the assert function and assert function termination operation

   [Diagnostic Information]
   < Program Diagnosis *** information of fail expression >
   file : File name expanded by __FILE__
   line : Line number expanded by __LINE__
   expression : Expression

   [Termination Operation]
   Same as the abort function calling.

2. Inspection character sets for isalnum, isalpha, iscntrl, islower, isprint, and isupper functions
   - isalnum: 0 to 9, a to z, or A to Z
   - isalpha: a to z or A to Z
   - iscntrl: \000 to \037, or \177
   - islower: a to z
   - isprint: \040 to \176
   - isupper: A to Z

3. Mathematical function return value upon definition area error occurrence
   - qNaN

4. Whether the mathematical function sets up the macro ERANGE value for errno upon underflow condition occurrence
   - ERANGE
   - The detectable result value must be +0 or –0.
   - The undetectable result value is undefined. It depends on the function.

5. When the second argument for the fmod function is 0, the definition area error must occur or the value 0 must be returned
   The definition area error must occur.

6. File buffering characteristics

   [Input File Buffering Characteristics]
   IOLBF, IOFBF: Full buffering.
   IONBF: No buffering.
APPENDIX

[Output File Buffering Characteristics]

IOFBF: Full buffering.
IOLBF: Line buffering.
IONBF: No buffering.

[Full Buffering]
Buffering is conducted using all the preset buffer areas.
When the input function is called at the time of input from a file, any data remaining in the
buffer is returned as the input from the file. If the buffer is emptied of data or does not
have sufficient data, the input from the file is received until the buffer is filled up and then
only the necessary amount is returned as the input.
At the time of output to a file, the output function writes into the buffer instead of
outputting into the file. When the buffer is filled up by the write operation, the buffer
outputs its entire contents to the file.

[Line Buffering]
Buffering is conducted for each output line.

[No Buffering]
File input/output is implemented in compliance with the input/output request made by
input/output function calling.
Unlike the other buffering operations, no data will be saved into the memory.

(7) Pointer size for %p format conversion
The fcc907s command handles the small model and medium model using 16 bits, and the
large model and compact model using 32 bits.

(8) %p format conversion output format for fprintf function
- Small Model/Medium Model:
  If the digit count is less than 4 in cases where the 4-digit hexadecimal notation is
  employed, leading 0s are added as needed. The alphabetical characters used are
  uppercased.
- Large Model/Compact Model:
  Same as for the small model except that the digit count is 8.

(9) Expansion of format conversion specification in fprintf, printf, sprintf, vfprintf, vprintf,
and vsprintf function
Expansion of %s and %n format conversion specification
- Small Model/Medium Model:
  It can be ordered that it be a pointer from which __far is qualified to the corresponding
  argument by specifying 'F'.
  [Example]
  ```
  #include <stdio.h>

  __far char a[] = "abc";
  main() { printf("%-16Fs\n", a); }
  ```
- Large Model/Compact Model:
  It can be ordered that it be a pointer from which __near is qualified to the corresponding
  argument by specifying 'N'.
  [Example]
#include <stdio.h>
__near char a[] = "abc";
main() { printf("%-16Ns\n", a); }

Expansion of %p format conversion specification

- **Small Model/Medium Model:**
  It can be ordered that it be a pointer from which __far is qualified to the corresponding argument by specifying 'l'.

  [Example]
  ```c
  #include <stdio.h>
  __far char a[] = "abc";
  main() { printf("%lp\n", a); }
  ```

- **Large Model/Compact Model:**
  It can be ordered that it be a pointer from which __near is qualified to the corresponding argument by specifying 'h'.

  [Example]
  ```c
  #include <stdio.h>
  __near char a[] = "abc";
  main() { printf("%hp\n", a); }
  ```

(10) **%p format conversion input format for fscanf function**

The fcc907s command adds leading 0s if the digit count is less than 4 (small model) or 8 (large model) when using upper- or lower-case alphabetic character-based hexadecimal notation. If the specified count of digits is exceeded, only the lower-order portion is valid.

(11) **Expansion of format conversion specification in fscanf, scanf, and sscanf function**

- **Small Model/Medium Model:**
  'F' can be specified for all the format conversion specification except %%%. It is shown that this 'F' is a pointer from which __far is qualified to the corresponding argument.

  [Example]
  ```c
  #include <stdio.h>
  __far int a;
  int b;
  main() { scanf("%Fd %d\n", &a, &b); }
  ```

- **Large Model/Compact Model:**
  'N' can be specified for all the format conversion specification except %%%. It is shown that this 'N' is a pointer from which __near is qualified to the corresponding argument.

  [Example]
  ```c
  #include <stdio.h>
  __near int a;
  int b;
  main() { scanf("%Nd %d\n", &a, &b); }
  ```
APPENDIX

(12) interpretation of a single "-" character appearing at a position other than the start and end of the scan-list relative to %[ format conversion

A string of consecutive characters beginning with the character placed to the left of "-" and ending with the character placed to the right of "-" is handled.

[Example]

%[a-c] is equal to %[abc].

(13) abort function operation relative to an open file

Closing takes place after flushing of all streams.

(14) Status returned by the exit function when the argument value is other than 0, EXIT_SUCCESS, and EXIT_FAILURE

The status to be returned is the same as for EXIT_FAILURE.

(15) Floating-point number limit values

- FLT_MAX 7F7F FFFF
- DBL_MAX 7FEF FFFF FFFF FFFF
- FLT_EPSILON 3400 0000
- DBL_EPSILON 3CB0 0000 0000 0000
- FLT_MIN 0080 0000
- DBL_MIN 0010 0000 0000 0000

(16) Limitations on setjmp function

The interrupt environment is not supported by the libraries. Therefore, the interrupt handler cannot achieve environment saving and the return to the interrupt handler cannot be made.

(17) Limitations on va_start macro

Do not use the following variable definitions for the va_start macro second argument.

- char type or unsigned char type (however, the pointer type for these types can be used.)
- Type having the register storage area class
- Function type
- Array type
- Structure type
- Union type
- Type different from the type derived from existing argument extension

(18) File types

Files that can be handled by the libraries are divided into two types; text files and binary files. The libraries treat the text files and binary files in the same manner except for the difference in the second argument of the open function called upon file opening.

When a binary file is specified, O_BINARY is added to the second argument of the open function. For the open function argument, see 8.5.1, “open Function”.
(19) **div_t type and ldiv_t type**

These are equal to the following structure.

```c
div_t:struct {
    int quot;
    int rem;
};
ldiv_t:struct {
    long int quot;
    long int rem;
};
```

(20) **abort function operations**

When the abort function is called, all the open output streams are flushed and then all the open streams are closed. Finally, the _abort function is called.

(21) **Maximum count of functions that can be registered by the atexit function**

Up to 32 functions can be registered.

(22) **exit function operations**

When the exit function is called, all the functions registered by the atexit function are called in the reverse order of registration, all the open output streams are flushed, and then all the open streams are closed.

Finally, the _exit function is called with the status value, which is delivered as the argument, retained. When the status value is 0 or EXIT_SUCCESS, it indicates successful termination. When the status value is EXIT_FAILURE, it indicates the unsuccessful termination.

(23) **The maximum number of files which can be opened at the same time according to fopen function**

The maximum number of files which can be opened at the same time is 20. When the limit is exceeded further and the file is opened, the fopen function returns **NULL**.
Appendix C  NOTES OF SIGNED DIVISION INSTRUCTION OF F^2MC-16LX CPU

Notes when F^2MC-16LX CPU is used are described.

Devices

All devices (Eva, OTP, FLASH, Mask) of F^2MC-16LX series:
All devices of QCM16LX core.

Notes in use

Normally remainder of the execution result for the signed division instruction ("DIV A,Ri" and "DIVW A,RWi") is set bank "00" area. But above devices set remainder bank (DTB/ADB/USB/SSB) area. When you use the signed division instruction, remainder is set at a bank area of the DTB/ADB/USB/SSB registers value.
Details are shown as follows.

Notes in use of "DIV A,Ri" and "DIVW A,RWi" instructions

The remainder of the execution result of the signed division instruction ("DIV A,Ri" and "DIVW A,RWi") is stored in the address (bit 0 to 15) which corresponds to the register of the instruction operand of memory bank area (bit 16 to 23) according to an undermentioned table. Therefore, please adjust the corresponding bank register to "00h" and use the "DIV A,Ri" and "DIVW A,RWi" instructions.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Bank Register</th>
<th>Address where the remainder is stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV A,R0</td>
<td>DTB</td>
<td>(DTB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIV A,R1</td>
<td>DTB</td>
<td>(DTB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIV A,R4</td>
<td>DTB</td>
<td>(DTB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIV A,R5</td>
<td>DTB</td>
<td>(DTB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIVW A,RW0</td>
<td>DTB</td>
<td>(DTB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIVW A,RW1</td>
<td>DTB</td>
<td>(DTB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIVW A,RW4</td>
<td>DTB</td>
<td>(DTB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIVW A,RW5</td>
<td>DTB</td>
<td>(DTB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIV A,R2</td>
<td>ADB</td>
<td>(ADB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIV A,R6</td>
<td>ADB</td>
<td>(ADB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIVW A,RW2</td>
<td>ADB</td>
<td>(ADB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIVW A,RW6</td>
<td>ADB</td>
<td>(ADB:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIV A,R3</td>
<td>USB *1</td>
<td>(USB *1:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIV A,R7</td>
<td>SSB *1</td>
<td>(USB *2:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIVW A,RW3</td>
<td></td>
<td>(USB *2:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
<tr>
<td>DIVW A,RW7</td>
<td></td>
<td>(USB *2:bit16 to 23)+(0180h+RPx10h+Ah:bit0 to 15)</td>
</tr>
</tbody>
</table>

*1 select by S bit of CCR register
*2 S bit of CCR register is 0
APPENDIX

When the value of the bank register is "00h", the remainder is stored in the register of the instruction operand. However, the remainder is stored in bank (DTB/ADB/USB/SSB) area, except when the value of the bank register is "00".

Example:

Case of DTB = 053h and RP = 003h
Address of R0 is 0180h+RP(3h)×10h+8h (R0 equivalent address) = 0001B8h. Bank register which used "DIV A,R0" is DTB which address is 053h.
Therefore, the remainder of the execution result of "DIV A,R0" is preserved in memory which address is 05301B8h.
(Please refer to the explanation for the general register of the manual for Ri and RWi.)

About avoiding the Notes

Please use this compiler and the assembler when you use the MB905XX series because the one that the function to replace the signed division instruction with an equivalent instructions was added will be changed in the compiler so as not to generate the signed division instruction to have the program evade the Notes and developed and be offered in the assembler as follows.

The kind which will be developed in the future will improve the Notes as MB904XX series.

Measures assembler : asm907a V03L04 or later  fasm907s V30L04(Rev.300004) or later
Measures compiler  : cc907 V02L06 or later  fcc907s  V30L02 or later

Moreover, this Notes can be avoided by having use in the F²MC-16L mode in a present compiler.

Supplementation explanation

About the influence on the program which has developed Notes

The Notes can be confirmed which the operation by Eva-device on a system. Therefore, the problem does not occur if a normal operation is confirmed in debugging though there is the signed division instruction in the program.

In the program development by the assembler:

(1) There is no problem if "DIV A,Ri" and "DIVW A,RWi" are not used.
(2) There is no problem if each bank register is "00h" though "DIV A,Ri" and "DIVW A,RWi" are used.
(3) The DIV instructions excluding this does not have the problem.

In the program development by C compiler:

(1) In small model and medium model, there is no problem when the bank register which ___far type qualified data and nor corresponds is used by "00h"(initial value).
   (In small model and medium model, C compiler does not change the value of each bank register initialized by the startup routine when there is no ___far type qualified data.)
(2) There is a possibility that "DIV A,R2", "DIV A,R6", "DIVW A,RW2", and the "DIVW A,RW6" instructions are influenced for either by ADB as follows even if the corresponding bank register is used by "00h"(initial value).
   - In small model and medium model, there is ___far type qualified data.
   - Compact model and large model are used.
(C compiler has the possibility to change the ADB register for the condition of (2))
However, there is no problem in the program if a normal operation is confirmed in debugging.
Appendix D  GUIDE TO CHANGING FUNCTION-CALL INTERFACE

If a function-call interface is changed, how to use selected registers and how to pass arguments of the function are also changed. Thus, the program will not operate properly if the existing function-call interface (called "stack argument passing") and a new function-call interface (called "register argument passing") are mixed. Here are the methods and precautions for changing the program to a new function-call interface.

#### New Function-call Interface

The F$^2$MC-16 Family SOFTUNE supports the new function-call interface ("register argument passing") to pass the arguments of the function by registers (RW0 and RW1) as well as the existing function-call interface ("stack argument passing") to pass the arguments of the function by the stack.

Using the function-call interface for "register argument passing", selected arguments are passed by registers, reducing code size and stack usage to improve the program execution speed.

However, the how to pass arguments and use registers differ between the "register argument passing" and the "stack argument passing." If objects with different function-call interfaces are mixed, the program will not operate properly. Workbench, Assembler, Linker, and Librarian are designed to output warning messages or error messages when detecting a contradiction concerning the function-call interface.

For details about the new function-call interface, refer to the following manuals:

- F$^2$MC-16 FAMILY SOFTUNE C COMPILER MANUAL
  - FUNCTION CALL INTERFACE (register argument passing)
  - Stack Frame (register argument passing)
  - Argument (register argument passing)
  - Argument Extension Format (register argument passing)
  - Calling Procedure (register argument passing)
  - Register (register argument passing)
  - Return Value (register argument passing)

- F$^2$MC-16 FAMILY SOFTUNE ASSEMBLER MANUAL
  - -rp, -Xrp
  - .REG_PASS instruction

- F$^2$MC-16 FAMILY SOFTUNE LINKAGE KIT MANUAL
  - Mixing of objects with different function-call interfaces in Linker
  - Option (-rp) to specify load module for "register argument passing"
APPENDIX

Option (-Xrp) to specify load module for "stack argument passing"
Mixing of objects with different function-call interfaces in Librarian
Option (-rp) to specify output of library for "register argument passing"
Option (-Xrp) to specify output of library for "stack argument passing"

○ F²MC-16 FAMILY SOFTUNE WORKBENCH OPERATION MANUAL

[Storing of Project]
[Setup project] - [General]

■ Changing of Function-call Interface

To change the function-call interface from "stack argument passing" to "register argument passing," use the following procedure:

Correct the Assembler source (e.g. startup .asm)
(See Correcting assembler source)

Is Workbench used for Build?

Yes

Change the function-call interface of the project to "register argument passing" and rebuild.
(See Changing by Workbench.)

No

Specify the -rp option for Compiler, Assembler, Linker and Librarian, and rebuild. (See Changing without using Workbench.)

Figure D-1 Changing of Function-call Interface

■ Details of Changing for Function-call Interface

○ Correcting assembler source

To change the function-call interface from "stack argument passing" to "register argument passing," use the following procedure to correct the assembler source:

(1) Description of .REG_PASS pseudo-instructions

When the -rp option is specified, Assembler outputs the object for "register argument passing."
However, only with the -rp option, Assembler cannot distinguish between the input assembler source for "register argument passing" and the input assembler source for "stack argument passing."

The .REG_PASS pseudo-instructions indicates the assembler source for "register argument passing."

When the -rp option is specified, if Assembler-assembles the assembler source where the .REG_PASS pseudo-instructions is not described, Assembler will output an error.

When changing the assembler source for "stack argument passing" to the assembler source for "register argument passing," describe the .REG_PASS pseudo-instructions.
The .REG_PASS pseudo-instructions can be described anywhere preceding the .END pseudo-instructions.

[Example of description for .REG_PASS pseudo-instructions]

```
.PROGRAM   _func
.TITLE     _func
; register argument passing
.REG_PASS
```

(2) Changing the how to pass arguments

If a function with arguments is defined in the assembler source and is called, change the how to pass arguments to “register argument passing.”

The same correction is also required when defining and calling the function with arguments by the assembler description function (asm statement) of C Compiler.

When changing the how to pass arguments to “register argument passing,” follow the tables in F2MC-16 Family SOFTUNE C COMPILER Manual "Arguments (register argument passing)."

[Correction example of definitions for functions with arguments]

(stack argument passing)           (register argument passing)

```
_func:                             _func:
    LINK    #0                    LINK    #0
    MOVW    A, @RW3+4            MOVW    RW4, @RW3+4
    ADDW    A, @RW3+6            MOVW    A, RW0
    ADDW    A, @RW3+8            ADDW    A, RW1
    MOVW    _var, A              ADDW    A, RW4
    :                             :                    MOVW    _var, A
    :                             :
```

[Correction example of calling functions with arguments]

(stack argument passing)           (register argument passing)

```
:                             :
    MOVW    A, #3              MOVW    A, #3
    PUSHW   A                  PUSHW   A
    MOVW    A, #2              MOVW    A, #2
    PUSHW   A                  MOVW    RW1, A
    MOVW    A, #1              MOVW    A, #1
    PUSHW   A                  MOVW    RW0, A
    CALL    _func              CALL    _func
    ADDSP    #6                POPW    AH
    :                             :
```
(Supplementary explanation)

Using the assembler source both for "stack argument passing" and for assembler source for "register argument passing"

Using the predefined macro "__REG_PASS__" create an assembler source both for "stack argument passing" and for "register argument passing".

When the predefined macro "__REG_PASS__" is assembled with the -rp option, set 1. When the predefined macro "__REG_PASS__" is assembled without the -rp option, set 0.

[Example of description for predefined macro "__REG_PASS__"]

```assembly
.PROGRAM _func
.TITLE _func
#if __REG_PASS__
; register argument passing
 .REG_PASS
#endif
```

>Note>

With the function-call interface for "register argument passing", return values from the functions of registers RW0 and RW1 are not guaranteed. Do not use these registers across function calls.

■ Changing by Workbench

Change the function-call interface of the project to "stack argument passing."

When the function-call interface for "register argument passing" is selected, the -rp option is specified automatically when Compiler, Assembler, Linker, and Librarian start up.
Appendix D  GUIDE TO CHANGING FUNCTION-CALL INTERFACE

(1) Changing the setting of the current project

Change the function-call interface of the project to "register argument passing" in the [Setup Project] dialog of the project.

![Figure D-2 Setup Dialog of Project](image)

(2) When changing the setting of a new project

In the [Create] dialog of the project, "stack argument passing" is selected by default.

Change the function-call interface to "register argument passing."

![Figure D-3 Create Dialog of Project](image)
<Note>

SOFTUNE REALOS supports only "stack argument passing." If the REALOS project is selected, only the function-call interface for "stack argument passing" can be selected.

There are some restrictions in the operation of the debugger when "register argument passing." is selected as a function call interface. These restrictions become the restrictions when the function to which it is called from the function with the argument, and the LINK/UNLINK instruction is not output by optimizing the compiler is debugged.

Operation of step-out (GO/RETURN command)
Since the parent function can no longer be obtained, execution of the program does not stop normally.

Operation of call stack (SHOW CALLS command)
Since the parent function can no longer be obtained, the subsequent parent functions cannot be displayed correctly.

Operation of the UP command
Since the parent function can no longer be obtained, the scope can not be correctly moved to the subsequent parent functions.

■ Changing without using Workbench

(1) Changing without using Workbench
When not using Workbench, specify the -rp option for Compiler, Assembler, Linker, and Librarian.

(2) Changing during compiling and assembling
Declare the prototypes of all functions. If a function whose prototype is not declared is called, the program is not guaranteed to operate.

The area where argument registers are saved is not concatenated with the area where parameters area passed by the stack. A program designed to expect such concatenation does not operate properly. If a program of this type exits, it must be corrected.

Similar to the conventional function-call interface, all arguments of the variable arguments are passed by the stack and need not to be corrected.

(3) Changing during linking and making library
The following C library is added for "register argument passing."
When using "register argument passing," link the following library with "/_rp."

```
lib/907/lib9*_rp.lib : 48 files
```

■ Warning and Error Messages

When a contradiction concerning the function-call interface is detected, Workbench, Assembler, Linker and Librarian output the following warning or error messages.

○ Workbench warning messages
When adding of project
I0227W
Projects with different function-call interfaces exist in the workspace.
Appendix D  GUIDE TO CHANGING FUNCTION-CALL INTERFACE

Objects created by the project with different function-call interfaces cannot be used together. 
Continue processing?

YES:  Add a project.
NO:  Do not add a project.

When setting dependencies between projects (sub projects)
I0228W
Function-call interfaces between projects are different.
Objects created by the project with different function-call interfaces cannot be used together. 
Continue processing?

YES:  Set the dependencies between projects.
NO:  Do not set the dependencies between projects.

When changing function-call interface
I0229W
When changing the function-call interface, the function-call interfaces of the assembler source, 
object, and library to be used must be standardized.
If the function-call interfaces are not standardized, the program is not guaranteed to operate. 
This change is applied to all project configurations in the project.
Change?

YES:  Change the function-call interface.
NO:  Do not change the function-call interface.

Assembly error message
E4718A
Different calling interface of a function.
This error message is outputted under the conditions shown in Appendix Table D-1.
Table D-1  Relationship between Description of .REG_PASS Pseudo-instructions and -rp Option

<table>
<thead>
<tr>
<th>Specifying of -rp option</th>
<th>Description of .REG_PASS pseudo-instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provided</td>
<td>Provided Normal combination for &quot;register argument passing&quot; No error is output.</td>
</tr>
<tr>
<td>Unprovided</td>
<td>The error (E4718A) is output. Normal combination for &quot;stack argument passing&quot; No error is output.</td>
</tr>
</tbody>
</table>
If these errors occur, correct the specifying of the -rp option or the description of the .REG_PASS pseudo-instructions for the assembler source, referring to Appendix Table D-1.

- **Linker error message**
  - E4313L
  - The module is different calling interface of a function (file name)
  - This error message is outputted under the conditions shown in Appendix Table D-2.

  **Table D-2  Relationship between Input Object and -rp Option**

<table>
<thead>
<tr>
<th>Specifying of -rp option</th>
<th>Function-call interface for input object</th>
<th>Function-call interface for input object and library to be edited</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;register argument passing&quot;</td>
<td>&quot;stack argument passing&quot;</td>
</tr>
<tr>
<td>Provided</td>
<td>Normal combination for &quot;register argument passing&quot; No error is output.</td>
<td>The error (E4313L) is output.</td>
</tr>
<tr>
<td>Unprovided</td>
<td>The error (E4313L) is output.</td>
<td>Normal combination for &quot;stack argument passing&quot; No error is output.</td>
</tr>
</tbody>
</table>

If these errors occur, correct the specifying of the -rp option or the function-call interface for the input object, referring to Appendix Table D-2.

- **Librarian error message**
  - E4410U
  - The module is different calling interface of a function (file name)
  - This error message is outputted under the conditions shown in Appendix Table D-3.

  **Table D-3  Relationship among Input Object, Library to Be Edited and -rp Option**

| Specifying of  
<table>
<thead>
<tr>
<th>-rp option</th>
<th>Function-call interface for input object and library to be edited</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;register argument passing&quot;</td>
</tr>
<tr>
<td>Provided</td>
<td>Normal combination for &quot;register argument passing&quot; No error is output.</td>
</tr>
<tr>
<td>Unprovided</td>
<td>The error (E4410U) is output.</td>
</tr>
</tbody>
</table>

If these errors occur, correct the specifying of the -rp option or the function-call interface for the input object and library to be edited, referring to Appendix Table D-3.

- **Note**
  - Objects with different function-call interfaces cannot be linked.
  - Fujitsu provides only the following object function-call interfaces for "stack argument passing."
  - Monitor Debugger
  - SOFTUNE REALOS
  - REALOS debug module (R_D_dbg.obj)
Appendix E  ERROR MESSAGE

The compiler displays the error message below.

■ Format of error message

<table>
<thead>
<tr>
<th>Error ID</th>
<th>Error message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1003D</td>
<td>Using the character not permitted as a module name.</td>
</tr>
</tbody>
</table>

[Explanation]
Characters other than alphabet (A to Z, a to z), numeric character (0 to 9), and underscore (_) are used as module name.

<table>
<thead>
<tr>
<th>Error ID</th>
<th>Error message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4002D</td>
<td>Illegal option-name</td>
</tr>
</tbody>
</table>

[Explanation]
The option name is wrong.
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4003D</td>
<td>Illegal value</td>
<td>The numerical value of the sub-option is not correct.</td>
</tr>
<tr>
<td>E4004D</td>
<td>Illegal sub-option</td>
<td>The sub-option is wrong.</td>
</tr>
<tr>
<td>E4005D</td>
<td>Illegal parameter description</td>
<td>A specified form of the option is wrong.</td>
</tr>
<tr>
<td>E4006D</td>
<td>Can not open option-file</td>
<td>The option file cannot be opened.</td>
</tr>
<tr>
<td>E4007D</td>
<td>Nested option-file exceeds 8</td>
<td>The nest level of the option file exceeded 8.</td>
</tr>
<tr>
<td>E4008D</td>
<td>Insufficient memory</td>
<td>More working memory area is needed.</td>
</tr>
<tr>
<td>E4009D</td>
<td>Illegal file name</td>
<td>The file name is wrong.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>E4010D</td>
<td>internal error : Illegal reserve number</td>
<td>internal error : Illegal reserve number</td>
</tr>
<tr>
<td>E4011D</td>
<td>Illegal section information</td>
<td>The section information is wrong.</td>
</tr>
<tr>
<td>E4012D</td>
<td>Illegal tool-item</td>
<td>The tool-item is wrong.</td>
</tr>
<tr>
<td>E4013D</td>
<td>Illegal sub-option</td>
<td>The sub-option is wrong.</td>
</tr>
<tr>
<td>E4014D</td>
<td>Illegal optimize level</td>
<td>The optimize level is wrong.</td>
</tr>
<tr>
<td>E4015D</td>
<td>internal error : Illegal prefix</td>
<td>internal error : Illegal prefix</td>
</tr>
<tr>
<td>E4016D</td>
<td>sub process call is failed</td>
<td>It failed in the start of the sub process.</td>
</tr>
</tbody>
</table>
## APPENDIX

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4017D</td>
<td>tool execute is failed</td>
<td>It failed in the call of the tool.</td>
</tr>
<tr>
<td>E4018D</td>
<td>help-file is not found</td>
<td>The Help file is not found.</td>
</tr>
<tr>
<td>E4019D</td>
<td>can't unlink the file</td>
<td>The file cannot be deleted.</td>
</tr>
<tr>
<td>E4020D</td>
<td>cannot process the C++ source file</td>
<td>The C++ source file cannot be compiled.</td>
</tr>
<tr>
<td>E4021D</td>
<td>option -cpu is not defined</td>
<td>The -cpu option is not specified.</td>
</tr>
<tr>
<td>E4022D</td>
<td>CPU information file not found</td>
<td>CPU information file is not found.</td>
</tr>
<tr>
<td>E4023D</td>
<td>CPU information not found</td>
<td>CPU information is not found.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>E4024D</td>
<td>too long file name.</td>
<td>The file name is too long.</td>
</tr>
<tr>
<td>E4025D</td>
<td>Can not open file</td>
<td>The file cannot be opened.</td>
</tr>
<tr>
<td>F9001D</td>
<td>internal error : can not find file</td>
<td>internal error : can not find file</td>
</tr>
<tr>
<td>F9002D</td>
<td>internal error : Illegal error-no</td>
<td>internal error : Illegal error-no</td>
</tr>
<tr>
<td>F9003D</td>
<td>Cannot create Ctrl-C Thread</td>
<td>Cannot create Ctrl-C Thread</td>
</tr>
<tr>
<td>F9004D</td>
<td>Internal error</td>
<td>Internal error</td>
</tr>
<tr>
<td>F9005D</td>
<td>Insufficient memory</td>
<td>More working memory area is needed.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>F9006D</td>
<td>Illegal CPU information file format</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] CPU information file format is not correct.</td>
<td></td>
</tr>
<tr>
<td>W1001P</td>
<td>#%R: empty expression: identifier required</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] No identifier is specified as a macro name of the #ifdef or #ifndef directive. Continues the compilation assuming that a macro name which was not defined is specified.</td>
<td></td>
</tr>
<tr>
<td>W1002P</td>
<td>#%R: invalid token specified after identifier: newline expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] An excess token is found after the macro name of the #ifdef or #ifndef directive. Continues the compilation neglecting the token after the macro name.</td>
<td></td>
</tr>
<tr>
<td>W1003P</td>
<td>#%R: expression expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] No constant expression to evaluate is specified to the #if or #elif directive. Continues the compilation assuming that 0 is specified.</td>
<td></td>
</tr>
<tr>
<td>W1004P</td>
<td>#%R: digit-sequence expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] In the #line directive, an excess character other than white space character is found after digits represent a line number. Continues the compilation assuming that digits represent a line number.</td>
<td></td>
</tr>
<tr>
<td>W1005P</td>
<td>too many arguments %d for macro `%s': %d expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The number of arguments specified in the macro expansion exceeds that of parameters in the definition. The excess is ignored.</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>W1006P</td>
<td>too few arguments %d for macro <code>\%s</code>: %d expected</td>
<td></td>
</tr>
</tbody>
</table>
|            | [Explanation] 
|            | The number of arguments specified in the macro expansion is less than that of parameters in the definition. Continues the compilation assuming that null string is specified for insufficient part. |
| W1007P     | invalid `,`: expression expected |
|            | [Explanation] 
|            | No argument is specified after the comma in the macro reference. Continues the compilation ignoring the comma. |
| W1008P     | unterminated character constant: `\"` expected |
|            | [Explanation] 
|            | The single quotation is missing to terminate the character constant. Continues the compilation assuming that it is specified at the end of the line or the end of the file. |
| W1009P     | unterminated string literal: `\"` expected |
|            | [Explanation] 
|            | The double quotation is missing to terminate the string literal. Continues the compilation assuming that it is specified at the end of the line or the end of the file. |
| W1010P     | -D option: `=` expected |
|            | [Explanation] 
|            | The `-D` option contains more than a macro name and the next character to the macro name is not `=`. Continues the compilation assuming that only the macro name is specified. |
| W1011P     | division by 0 |
|            | [Explanation] 
|            | The constant operation in the condition expression contains the division by 0. Continues the compilation assuming that the result of the operation is 0. |
### W1012P
trigraph sequence `\?!%c' replaced with `%c'

[Explanation]
The replacement of the trigraph sequence is made. The resulting character is valid.

### W1013P
`$' in identifier

[Explanation]
The `$' is contained in an identifier. It is regarded as a part of the identifier.

[Note]
This message is not used.

### W1014P
parameter `%s' of macro `%s' in string literal may be replaced

[Explanation]
An identifier in the string literal can be regarded as a parameter of a macro and macro replacement may be made. Continues the compilation making the macro replacement.

### W1015P
parameter `%s' of macro `%s' in character constant may be replaced

[Explanation]
An identifier in the character constant can be regarded as a parameter of macro and macro replacement may be made. Continues the compilation making the macro replacement.

### W1016P
previous defined at "%s", line %d

[Explanation]
This is a supplemental message of W1026P.

### W1017P
previous defined at command line

[Explanation]
This is a supplemental message of W1026P.
W1018P  this macro is predefined macro

[Explanation]
This is a supplemental message of W1026P.

W1019P  the only white space allowed within pp directive is space and horizontal tab

[Explanation]
A horizontal tab or a form feed character is found in the preprocessor directive line. Continues the compilation regarding it as white space character.

W1020P  ANSI C extension: #%R specified

[Explanation]
The preprocessor directive is not specified in the ANSI C. Continues the compilation regarding it as the ANSI C extension.

W1021P  %d trigraph sequence replaced in this file

[Explanation]
An information of the trigraph sequence replacement.

[Note]
This message is not used.

W1022P  EOF in comment

[Explanation]
The "*/" is missing to terminate the comment. Continues the compilation assuming that "*/" is at the end of the file.

W1023P  invalid character \0%o

[Explanation]
The character is internal use only in the preprocess. Continues the compilation assuming it as a white space.
## Comment replaced with `##` operator

<table>
<thead>
<tr>
<th>W1024P</th>
<th>[Explanation] An old style specification is applied which regards &quot;a/**/b&quot; as &quot;a##b&quot;. Continues the compilation making it valid.</th>
</tr>
</thead>
</table>

## `%R: cannot define macro `%s'`  

<table>
<thead>
<tr>
<th>W1025P</th>
<th>[Explanation] The identifier is reserved by the preprocessor and user cannot redefine it as macro name. Continues the compilation making the macro definition invalid.</th>
</tr>
</thead>
</table>

## `%R: macro `%s' redefined`  

<table>
<thead>
<tr>
<th>W1026P</th>
<th>[Explanation] The identifier is already defined as macro differently. Continues the compilation making the preceding definition valid.</th>
</tr>
</thead>
</table>

## `%R: invalid token specified after identifier: newline expected`  

<table>
<thead>
<tr>
<th>W1027P</th>
<th>[Explanation] An excess token is found after the identifier in the #undef directive. Continues the compilation neglecting the token.</th>
</tr>
</thead>
</table>

## `%R: cannot undefine macro `%s'`  

<table>
<thead>
<tr>
<th>W1028P</th>
<th>[Explanation] The macro name specified in identifier of the #undef directive cannot be canceled. Continues the compilation making the #undef directive invalid.</th>
</tr>
</thead>
</table>

## Invalid token after #else`  

<table>
<thead>
<tr>
<th>W1029P</th>
<th>[Explanation] An excess token is found in the #else directive. Continues the compilation neglecting it.</th>
</tr>
</thead>
</table>

## Invalid token after #endif`  

<table>
<thead>
<tr>
<th>W1030P</th>
<th>[Explanation] An excess token is found in the #endif directive. Continues the compilation neglecting it.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Code</td>
<td>Error Message</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>W1031P</td>
<td>invalid token follows at the end of #include directive line</td>
</tr>
<tr>
<td>W1032P</td>
<td>#include: cannot find file &quot;%s&quot;</td>
</tr>
<tr>
<td>W1033P</td>
<td>#include: cannot find file &lt;%s&gt;</td>
</tr>
<tr>
<td>W1034P</td>
<td>#%R: syntax error: identifier expected</td>
</tr>
<tr>
<td>W1035P</td>
<td>#%R: <code>(</code> required</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>W1036P</td>
<td>#%R: syntax error: <code>.</code> or <code>)</code> expected</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td>W1037P</td>
<td>#%R: newline expected: invalid token follows after <code>)</code></td>
</tr>
<tr>
<td>W1038P</td>
<td>#%R: filename expected</td>
</tr>
<tr>
<td>W1039P</td>
<td>#%R: line number 0 specified</td>
</tr>
<tr>
<td>W1040P</td>
<td>#%R: specified line number is greater than 32767</td>
</tr>
<tr>
<td>W1041P</td>
<td>macro <code>%s</code> recursion</td>
</tr>
</tbody>
</table>
Appendix E  ERROR MESSAGE

W1042P  #%R: invalid directive in macro parentheses

[Explanation]
The preprocessor directive written within argument expansion of function format macro is invalid. Continues the compilation neglecting the directive.

[Note]
This message is not used.

W1043P  #%R: directive used in macro parentheses

[Explanation]
The preprocessor directive is found within argument expansion of function format macro. Continues the compilation making the directive valid.

W1044P  invalid directive

[Explanation]
The token which follows the `#' cannot be recognized as a preprocessor directive. Continues the compilation neglecting the directive.

W1045P  unknown option -X%c

[Explanation]
The option for the language specification is specified badly. Continues the compilation assuming that the option were not specified.

W1046P  unknown option -X

[Explanation]
The option for the language specification is specified badly. Continues the compilation neglecting the option.

W1047P  too many parameters on command line

[Explanation]
An option or a file name is specified after the output file name in the command line. Continues the compilation neglecting the excess argument.
APPENDIX

W1048P unknown option %s

[Explanation]
Unknown option is detected.

[Note]
This message is not used.

W1049P invalid option %s

[Explanation]
Unknown option is detected.

[Note]
This message is not used.

W1050P invalid digit postfix expression

[Explanation]
The `L' or the `l' is specified more than once as the postfix of the integral constant in the condition expression. Continues the compilation neglecting the postfix.

W1051P integer constant out of range

[Explanation]
The specified integral constant cannot be expressed internally.

[Note]
This message is not used.

W1052P character constant too long

[Explanation]
The wide-character constant in the condition expression is too long. Only the first character is valid.
W1053P newline in string literal

[Explanation]
The double quotation is missing to terminate the string literal.

[Note]
This message is not used.

W1054P numeric octal constant contains ‘8’ or ‘9’

[Explanation]
‘8’ or ‘9’ is found in the octal integer constant in the condition expression. Continues the compilation regarding ‘8’ as ‘010’, ‘9’ as ‘011’.

W1055P invalid character ‘\%o’ (octal)

[Explanation]
An invalid character is found in the source program. Continues the compilation neglecting the character.

W1056P alert escape sequence is specified

[Explanation]
The ‘\a’ is not the alert escape sequence in the old specification. Continues the compilation regarding the ‘\a’ as the alert escape sequence.

W1057P escape sequence ‘\x’ is specified

[Explanation]
In the old specification, ‘\x’ cannot be used to represent a hexadecimal number. Continues the compilation regarding ‘\x’ as ‘x’ and a hexadecimal digit as a simple character.

W1058P ‘$’ character in identifier

[Explanation]
The ‘$’ is contained in an identifier. It is regarded as a part of the identifier.

[Note]
This message is not used.
APPENDIX

W1059P  `long long' integer constant is used

[Explanation]
The `L' or the `l' is specified more than once as the postfix of the integral constant in the
condition expression. Continues the compilation neglecting the postfix.

W1060P  unterminated filename

[Explanation]
The last double quotation is missing to enclose the file name in the #line number directive.

[Note]
This message is not used.

W1061P  too long hexadecimal escape sequence

[Explanation]
The hexadecimal number by the escape sequence `\x' is larger in size than a character.
Continues the compilation making only the lowest byte or wide-character valid.

W1062P  `*/' exists outside of comment

[Explanation]
The `*/' is found in the condition expression. It is regarded simply as `*' and `/'.

W1063P  hexadecimal escape sequence has no digit value

[Explanation]
No hexadecimal digit follows the escape sequence `\x'. Continues the compilation assuming that the `\x0' is specified.

W1064P  unknown escape sequence `\%c'

[Explanation]
A character which not specified as an escape sequence in the ANSI C is found after the `\'.
Continues the compilation neglecting the `\'.
Appendix E  ERROR MESSAGE

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1065P</td>
<td>too large integer constant for radix %d</td>
<td>The digit-sequence is too long and surely overflow before converting into the internal format. Continues the compilation assuming that 0 is specified.</td>
</tr>
<tr>
<td>W1066P</td>
<td>escape sequence does not fit in range of character</td>
<td>The result of the escape sequence cannot be represented by a byte (ex. \400). Continues the compilation making only the lowest byte valid.</td>
</tr>
<tr>
<td>W1067P</td>
<td>escape sequence does not fit in range of wide character</td>
<td>The result of the escape sequence cannot be represented by four bytes. Continues the compilation making only the lowest wide character valid.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1068P</td>
<td>too large integer constant `%s' for radix %d</td>
<td>The digit-sequence is too long and surely overflow before converting into the internal format. Continues the compilation assuming that 0 is specified.</td>
</tr>
<tr>
<td>W1069P</td>
<td>exceed the maximum length of octal escape sequence</td>
<td>The octal number written in the escape sequence form consists of more than four digits.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1070P</td>
<td>unknown escape sequence `\%o'</td>
<td>A character which is not specified as an escape sequence in the ANSI C is found after the <code>\</code>. Continues the compilation neglecting the <code>\</code>.</td>
</tr>
</tbody>
</table>
## APPENDIX

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1071P</td>
<td>too long multi-character character constant</td>
</tr>
</tbody>
</table>

**[Explanation]**
A character constant which consists of more than five characters is found. Continues the compilation making the first four characters valid.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1072P</td>
<td>multi-character character constant specified</td>
</tr>
</tbody>
</table>

**[Explanation]**
A character constant which consists of more than two characters is found. Continues the compilation making it valid.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1073P</td>
<td>cannot open compiler message file &quot;%s&quot;</td>
</tr>
</tbody>
</table>

**[Explanation]**
A message built in the preprocessor is issued.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1074P</td>
<td>assertion is ANSI C extension</td>
</tr>
</tbody>
</table>

**[Explanation]**
The assertion is not the ANSI C standard. Continues the compilation evaluating it as an assertion.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1075P</td>
<td>%#R: invalid token: identifier required</td>
</tr>
</tbody>
</table>

**[Explanation]**
A token which is not an identifier is specified as a macro name of the #ifdef or #ifndef directive. Continues the compilation assuming that an undefined macro names are specified.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1076P</td>
<td>%#R: too large decimal constant</td>
</tr>
</tbody>
</table>

**[Explanation]**
A number larger than the max value of the long type, 2147483647, is specified as a line number. Continues the compilation regarding it as a number of the unsigned long type.
### Appendix E  ERROR MESSAGE

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1077P</td>
<td>#operator used in macro replacement of non function-like macro</td>
<td>The function of the <code>#</code> operator is converting parameters of a function-like macro into string literals. It replaces as normal <code>#</code> character.</td>
<td></td>
</tr>
<tr>
<td>W1078P</td>
<td>unterminated string literal: EOF in string literal</td>
<td>The last double quotation is missing to enclose the string literal.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1079P</td>
<td>unterminated character constant: EOF in character constant</td>
<td>The last single quotation is missing to enclose the character constant.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1080P</td>
<td>unterminated string literal: newline in string literal</td>
<td>The last double quotation is missing to enclose the string literal.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1081P</td>
<td>unterminated character constant: newline in character constant</td>
<td>The last single quotation is missing to enclose the character constant.</td>
<td>This message is not used.</td>
</tr>
</tbody>
</table>
APPENDIX

W1082P | too long identifier, truncated to `%s'

[Explanation]
The identifier is too long.

[Note]
This message is not used.

W1083P | cannot concatenate character string literal and wide character string literal: assumed as character string literal

[Explanation]
A wide character string literal follows a character string literal.

[Note]
This message is not used.

W1084P | cannot concatenate wide character string literal and character string literal: assumed as wide character string literal

[Explanation]
A character string literal follows a wide character string literal.

[Note]
This message is not used.

W1085P | decimal integer constant is too large

[Explanation]
The decimal integer constant is too large.

[Note]
This message is not used.

W1086P | #pragma asm: syntax error: `#pragma endasm` is not specified

[Explanation]
The #pragma endasm for the #pragma asm is missing. Continues the compilation assuming that it is specified at the end of the file.
W1087P  #include: filename too long: file `%s'

[Explanation]
The full path file name in the #include directive is too long. Only the first 255 bytes are valid.

W1088P  #include: filename too long: include path `%s', file `%s'

[Explanation]
The path name of the file name specified in the #include directive with its directory name specified by the -I option is too long. Only the first 255 bytes are valid.

W1089P  #%R: invalid token after `%s': newline expected

[Explanation]
Only a newline is allowed after the "#pragma asm" or the "#pragma endasm". The indicated token is ignored.

W1090P  <<note:#warning directive displays following literals written in the source file as they are>>

[Explanation]
The "#warning" is recognized.

E4001P  mismatch #if-#endif

[Explanation]
Numbers of #if and #endif in a file are not equal. Mostly #endif is missing.

E4002P  cannot get current time (time())

[Explanation]
System call for the current time failed, and values of __DATE__ and __TIME__ macros are not safe.

E4003P  unacceptable token in constant expression

[Explanation]
There is a token which cannot be recognized as a constant expression written in the #if or other directive in condition expression.
APPENDIX

E4004P  pp-token required before ## operator

[Explanation]
Though ## operator concatenates preceding and following tokens, there is no preceding token.

E4005P  pp-token required after ## operator

[Explanation]
Though ## operator concatenates preceding and following tokens, there is no following token.

E4006P  identifier required after # operator

[Explanation]
Though # operator makes parameters of a function-like macro into literal strings, no identifier follows # operator as a macro parameter.

E4007P  macro parameter required after # operator

[Explanation]
Though # operator makes parameters of a function-like macro into literal strings, the identifier which follows the # operator is not a macro parameter.

E4008P  assertion: `)' expected

[Explanation]
No assertion specifier name is specified in referring an assertion predicate name.

E4009P  assertion: identifier required after `#'

[Explanation]
No assertion predicate name is specified in referring an assertion predicate name.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4010P</td>
<td>assertion: <code>(</code> required after <code>#identifier</code></td>
<td>The <code>(</code> is missing to enclose the assertion specifier in referring an assertion predicate name.</td>
</tr>
<tr>
<td>E4011P</td>
<td>assertion: empty within parentheses</td>
<td>No identifier is specified as an assertion specifier in referring an assertion predicate name.</td>
</tr>
<tr>
<td>E4012P</td>
<td>identifier required after <code>defined</code> operator</td>
<td>No identifier is specified after the defined operator.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4013P</td>
<td>assertion: <code>)</code> required after `#identifier(...'</td>
<td>The <code>)</code> is missing to enclose the assertion specifier in referring an assertion predicate name.</td>
</tr>
<tr>
<td>E4014P</td>
<td>write error (fwrite())</td>
<td>An error occurred while writing the output of the preprocessor to a file.</td>
</tr>
<tr>
<td>E4015P</td>
<td>`#%R: duplicate formal parameter</td>
<td>Some parameters specified in the function-like macro definition by the #define directive have the same spelling.</td>
</tr>
</tbody>
</table>
### APPENDIX

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4016P</td>
<td><code>%R: parameter syntax error: </code>,’ or <code>)’ expected</code></td>
</tr>
</tbody>
</table>

**[Explanation]**

In the function-like macro definition by the `#define` directive, a separator `,’ or a terminator `)’ is missing in the parameter list.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4017P</td>
<td><code>%R: invalid parameter: identifier required</code></td>
</tr>
</tbody>
</table>

**[Explanation]**

In the function-like macro definition by the `#define` directive, a parameter which is not an identifier is specified.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4018P</td>
<td><code>%R: invalid token: identifier required</code></td>
</tr>
</tbody>
</table>

**[Explanation]**

The macro name of a `#define` or `#undef` directive or a `#assert` predicate name should be an identifier.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4019P</td>
<td><code>%R: empty macro name: identifier required</code></td>
</tr>
</tbody>
</table>

**[Explanation]**

No identifier is specified as a macro name in the `#define` or `#undef` directive.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4020P</td>
<td>unexpected <code>#elif</code></td>
</tr>
</tbody>
</table>

**[Explanation]**

The `#if` for the `#elif` directive is missing.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4021P</td>
<td><code>#elif</code> follows after <code>#else</code></td>
</tr>
</tbody>
</table>

**[Explanation]**

The `#elif` directive appears after the `#else` directive. The `#else` directive should appear after the `#elif` directive.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4022P</td>
<td>unexpected #else</td>
<td>The #if for the #else directive is missing.</td>
</tr>
<tr>
<td>E4023P</td>
<td>#else follows #else</td>
<td>The #else directive appears after the #else directive. In a #if closes, the #else directive should appear only once.</td>
</tr>
<tr>
<td>E4024P</td>
<td>unexpected #endif</td>
<td>The #if for the #endif directive is missing.</td>
</tr>
<tr>
<td>E4025P</td>
<td>#include: empty directive line</td>
<td>No file name is specified in the #include directive.</td>
</tr>
<tr>
<td>E4026P</td>
<td>#include: invalid character: <code>&lt;' or </code>&quot;' expected</td>
<td>The file name in the #include directive does not begin with <code>&lt;' nor </code>&quot;`.</td>
</tr>
<tr>
<td>E4027P</td>
<td>#include: unterminated filename: `%' expected</td>
<td>The file name in the #include directive does not end with <code>%' nor </code>&quot;`.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>E4028P</td>
<td>#include: empty filename</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] No file name is specified within &lt;&gt; or &quot;&quot; in the #include directive.</td>
<td></td>
</tr>
<tr>
<td>E4029P</td>
<td>#%R: digit-sequence expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] No digit-sequence is specified for line number in the #line directive.</td>
<td></td>
</tr>
<tr>
<td>E4030P</td>
<td>#%R: too large decimal constant</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The decimal digit-sequence for line number in the #line directive is too long.</td>
<td></td>
</tr>
<tr>
<td>E4031P</td>
<td>#%R: `&quot;' missing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The character string expressed the file name specified in the #line directive does not end with <code>&quot;</code>.</td>
<td></td>
</tr>
<tr>
<td>E4032P</td>
<td>defined operator: invalid token: identifier expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The operand within the parenthesis in the defined operator is not an identifier.</td>
<td></td>
</tr>
<tr>
<td>E4033P</td>
<td>defined operator: `)' expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] Though the operand of the defined operator begins with <code>(', it does not end with </code>)`.</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>E4034P</td>
<td>defined operator: identifier or `(' expected</td>
<td>A token other than identifier nor `(' appears just after the defined operator.</td>
</tr>
<tr>
<td>E4035P</td>
<td>unterminated macro <code>%s</code>: <code>)</code> expected</td>
<td>The <code>)</code> is missing to enclose arguments in the function-like macro reference.</td>
</tr>
<tr>
<td>E4036P</td>
<td>unterminated character constant: <code>&quot;</code> expected</td>
<td>The single quotation is missing to terminate the character constant.</td>
</tr>
<tr>
<td>E4037P</td>
<td>unterminated string literal: <code>\&quot;</code> expected</td>
<td>The double quotation is missing to terminate the string literal.</td>
</tr>
<tr>
<td>E4038P</td>
<td>#include: cannot find file <code>%s</code></td>
<td>The private include file is not found.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>E4039P</td>
<td>#include: cannot find file <code>&lt;%s&gt;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The standard include file is not found.</td>
<td></td>
</tr>
<tr>
<td>E4040P</td>
<td><code>%R: syntax error: identifier expected</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] No identifier is specified for a parameter between <code>,</code> and <code>)</code> in the function-like macro definition by <code>#define</code> directive.</td>
<td></td>
</tr>
<tr>
<td>E4041P</td>
<td><code>%R: empty in parentheses</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] No identifier is written within the parenthesis enclosing an assertion specifier in the <code>#assert</code> or <code>#unassert</code> directive.</td>
<td></td>
</tr>
<tr>
<td>E4042P</td>
<td>unknown directive</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] Unknown preprocessor directive is specified.</td>
<td></td>
</tr>
<tr>
<td>E4043P</td>
<td><code>%R: syntax error: </code>)<code> expected</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The <code>)</code> is missing to enclose the assertion specifier in the <code>#assert</code> or <code>#unassert</code> directive.</td>
<td></td>
</tr>
<tr>
<td>E4044P</td>
<td><code>%R: token sequence expected</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] Nothing is written after the <code>(</code> which leads to an assertion specifier in the <code>#assert</code> or <code>#unassert</code> directive.</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E4045P</td>
<td>too nested include file</td>
<td>The nest level of include exceeds the limit of this system, 64.</td>
</tr>
<tr>
<td>E4046P</td>
<td>unterminated string literal: `&quot;' expected</td>
<td>The `&quot;' is missing to terminate the string literal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Note]                        This message is not used.</td>
</tr>
<tr>
<td>E4047P</td>
<td>EOF in comment</td>
<td>The <code>/*</code> is missing to terminate the comment written in the condition expression.</td>
</tr>
<tr>
<td>E4048P</td>
<td>EOF in string literal</td>
<td>The `&quot;' is missing to terminate the string literal in the condition expression.</td>
</tr>
<tr>
<td>E4049P</td>
<td>sorry, internal limitation: quoted character too long</td>
<td>The length of the character constant written in condition expression or the hexadecimal beginning with &quot;\x&quot; in the string literal exceeds the limit of this system, 4028.</td>
</tr>
<tr>
<td>E4050P</td>
<td>character constant has no character expression</td>
<td>No character is written in the character constant in the condition expression.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E4051P</td>
<td>unterminated character constant: newline in character constant</td>
<td>Newline is found in the character constant.</td>
</tr>
<tr>
<td>E4052P</td>
<td>unterminated character constant: EOF in character constant</td>
<td>The single quotation is missing to terminate the character constant.</td>
</tr>
<tr>
<td>E4053P</td>
<td>too many postfix characters <code>\%c</code> for constant</td>
<td>The postfix of the numerical constant in the condition expression is not specified correctly.</td>
</tr>
<tr>
<td>E4054P</td>
<td>numeric octal constant contains invalid character</td>
<td>A character which is not an octal digit is used in the octal constant.</td>
</tr>
<tr>
<td>E4055P</td>
<td>binary constant cannot be floating point constant</td>
<td>Binary constant beginning with <code>\0b</code> or <code>\0B</code> cannot be floating point constant.</td>
</tr>
</tbody>
</table>
E4056P invalid postfix character `\%c' after integer constant

[Explanation]
An alphabet which cannot be recognized as a postfix is specified at the end of the integral constant in the condition expression.

E4057P invalid postfix character `\%c' for radix %d

[Explanation]
A character which, not an alphabet, cannot be recognized as a postfix is specified at the end of the integral constant in the condition expression.

E4058P no digits of floating exponent part

[Explanation]
No digit is specified for the exponent part of the floating point number in the condition expression.

E4059P hexadecimal constant cannot be floating point constant

[Explanation]
The `.' is found in the hexadecimal constant in the condition expression.

E4060P invalid postfix character `\%c' after floating point constant

[Explanation]
An invalid character for postfix of floating point number is specified at the end of the floating point constant in the condition expression.

E4061P invalid token: `..'

[Explanation]
An invalid token `..' is found in the condition expression.
### APPENDIX

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4062P</td>
<td>integer constant out of range</td>
<td>The integral constant cannot be expressed internally.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4063P</td>
<td>invalid character <code>\%c</code></td>
<td>An invalid character is found in the condition expression.</td>
<td></td>
</tr>
<tr>
<td>E4064P</td>
<td>invalid binary constant</td>
<td>The first digit of the binary constant beginning with <code>0b</code> or <code>0B</code> is not 0 nor 1.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4065P</td>
<td>invalid hexadecimal constant</td>
<td>The first digit of the hexadecimal constant beginning with <code>0x</code> or <code>0X</code> is not a hexadecimal digit.</td>
<td></td>
</tr>
<tr>
<td>E4066P</td>
<td>invalid multibyte character constant</td>
<td>There is an invalid multi-byte character which cannot be recognized as a character code written in the wide-character constant in the condition expression.</td>
<td></td>
</tr>
</tbody>
</table>
Appendix E  ERROR MESSAGE

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4067P</td>
<td>invalid multibyte string literal</td>
<td>[Explanation] There is an invalid multi-byte character which cannot be recognized as a character code in the wide-character string literal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>E4068P</td>
<td>invalid character `%o' (octal)</td>
<td>[Explanation] An invalid character is found in the condition expression.</td>
</tr>
<tr>
<td>E4069P</td>
<td>%s near wide character string constant</td>
<td>[Explanation] This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the wide-character string literal.</td>
</tr>
<tr>
<td>E4070P</td>
<td>%s near wide character constant</td>
<td>[Explanation] This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the wide-character constant.</td>
</tr>
<tr>
<td>E4071P</td>
<td>%s near `%s'</td>
<td>[Explanation] This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the specified character.</td>
</tr>
</tbody>
</table>
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4072P</td>
<td>%s detected</td>
<td>This is a supplemental message for syntax error in a condition expression: A syntax error occurred.</td>
</tr>
<tr>
<td>E4073P</td>
<td>%s near character constant</td>
<td>This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the character constant.</td>
</tr>
<tr>
<td>E4074P</td>
<td>%s near string constant</td>
<td>This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the string literal.</td>
</tr>
<tr>
<td>E4075P</td>
<td>cannot quote EOF</td>
<td>The EOF is found just after the backslash.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>E4076P</td>
<td>invalid `' in input</td>
<td>The backslash is found in the condition expression.</td>
</tr>
<tr>
<td>E4077P</td>
<td>&lt;&lt;note:#error directive displays following literals written in the source file as they are&gt;&gt;</td>
<td>The &quot;#error&quot; is recognized.</td>
</tr>
</tbody>
</table>
## Appendix E  ERROR MESSAGE

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4078P</td>
<td>-default filename expected</td>
</tr>
</tbody>
</table>

[Explanation]
No sub-parameter is set to the "-default" option to specify a file name.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4079P</td>
<td>-predefine filename expected</td>
</tr>
</tbody>
</table>

[Explanation]
No sub-parameter is set to the "-predefine" option to specify a file name.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4080P</td>
<td>invalid <code>$</code> in input</td>
</tr>
</tbody>
</table>

[Explanation]
An invalid character `$` is found in the condition expression.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4081P</td>
<td>invalid option %s</td>
</tr>
</tbody>
</table>

[Explanation]
No sub-parameter is set to the "-X" option to specify language level specification.

[Note]
This message is not used.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4082P</td>
<td>write error (timeout)</td>
</tr>
</tbody>
</table>

[Explanation]
An error occurred while writing the output of the preprocessor to a file.

[Note]
This message is not used.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4083P</td>
<td>write error (select())</td>
</tr>
</tbody>
</table>

**[Explanation]**
An error occurred while writing the output of the preprocessor to a file.

**[Note]**
This message is not used.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4084P</td>
<td>write error (write())</td>
</tr>
</tbody>
</table>

**[Explanation]**
An error occurred while writing the output of the preprocessor to a file.

**[Note]**
This message is not used.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4085P</td>
<td>unknown # directive</td>
</tr>
</tbody>
</table>

**[Explanation]**
An unknown preprocessor directive is specified.

**[Note]**
This message is not used.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4086P</td>
<td>invalid character `%o' (octal) in # directive line</td>
</tr>
</tbody>
</table>

**[Explanation]**
An unknown character is found in the preprocessor directive.

**[Note]**
This message is not used.
<table>
<thead>
<tr>
<th>Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4087P</td>
<td>cannot open file &quot;%s&quot;</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The file specified in the &quot;-Hf&quot; option cannot be opened for writing.</td>
</tr>
<tr>
<td>F9001P</td>
<td>sorry, internal limitation: too nested input</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Too many changes are made on the source file by macro substitution to process.</td>
</tr>
<tr>
<td>F9002P</td>
<td>detected too many errors to terminate compilation</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Too many errors are detected to continue.</td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>F9003P</td>
<td>Broken pipe</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Pipe is broken.</td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>F9004P</td>
<td>too many comments</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The number of comment exceeds the limit of this system, 16777215.</td>
</tr>
<tr>
<td>F9005P</td>
<td>cannot get file mode (fstat)</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Cannot get the file information to read.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>F9006P</td>
<td>fread</td>
</tr>
<tr>
<td>F9007P</td>
<td>invalid file mode</td>
</tr>
<tr>
<td>F9008P</td>
<td>illegal message file</td>
</tr>
<tr>
<td>F9009P</td>
<td>virtual memory exhausted</td>
</tr>
<tr>
<td>I0001C</td>
<td>previous declaration of <code>\%D': </code>%s', line %d</td>
</tr>
<tr>
<td>I0002C</td>
<td>on empty parameter declaration of function declaration</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>I0003C</td>
<td>on formal parameter declaration of function definition</td>
</tr>
<tr>
<td>I0004C</td>
<td>using ellipsis terminator on empty parameter declaration</td>
</tr>
<tr>
<td>I0005C</td>
<td>incompatible types between <code>%T</code> and <code>%T</code></td>
</tr>
<tr>
<td>I0006C</td>
<td><code>#pragma echo</code> is not available on your system</td>
</tr>
<tr>
<td>I0007C</td>
<td>``%D<code>is built-in symbol:</code>%T`</td>
</tr>
<tr>
<td>W1006C</td>
<td>pointer to function specified</td>
</tr>
</tbody>
</table>
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1007C</td>
<td>empty declaration</td>
<td>A semicolon alone is written outside the function. A declaration is to be written outside the function. Continues the compilation neglecting the semicolon.</td>
</tr>
<tr>
<td>W1008C</td>
<td>unknown size of incomplete type</td>
<td>A tag name of the struct, union or enumeration without a body defined is specified in an operand of the sizeof operator. An operand of the sizeof operator must be of a complete type.</td>
</tr>
<tr>
<td>W1009C</td>
<td>unknown size of incomplete type</td>
<td>An array type whose size is unknown is specified in an operand of the sizeof operator. An operand of the sizeof operator must be of a complete type.</td>
</tr>
<tr>
<td>W1010C</td>
<td>unknown size of function type</td>
<td>Attempted to get the size of the function type. It's impossible, so 1 is assumed.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is sub message of W1124C,W1125C.</td>
</tr>
<tr>
<td>W1011C</td>
<td>unknown size</td>
<td>Attempted to get the size of the void type. It's impossible, so 1 is assumed.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is sub message of W1124C,W1125C.</td>
</tr>
</tbody>
</table>
W1012C invalid null subscript of array type

[Explanation]
Attempted to get the size of the array type whose length is unknown. Zero is assumed.

[Note]
This message is sub message of W1124C, W1125C.

W1013C invalid variable subscript of array

[Explanation]
The subscript of an array type specified in the operand of the sizeof operator is not an integral constant. The size of an operand for the sizeof operator must be determined at compile time.

[Note]
This message is not used.

W1014C identifier without type in function parameter declaration

[Explanation]
No type is specified in a parameter declaration of a function.

[Note]
This message is not used.

W1015C both return nothing and return value are used in function `%D`

[Explanation]
Though several return statements are written in a function, some return values and others mix. Unless the return type of a function is the void type, all the return statement must return a value.
Continue the compilation assuming that an unknown value is returned if no return value is specified.
<table>
<thead>
<tr>
<th>Message ID</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1016C</td>
<td>%Z: array dimension is variable</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Only an integral constant can be specified for a subscript of an array in a declaration.</td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>W1017C</td>
<td>%Z: brace-enclosed list of initializers expected for aggregate type</td>
</tr>
<tr>
<td></td>
<td>[Explanation] An initial value for a struct, union or array is not enclosed with &quot;{}&quot;. Continue the compilation making it up.</td>
</tr>
<tr>
<td>W1018C</td>
<td>%Z: too long %s literal (%u) for array: `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Too long string literal is specified for the initial value to initialize an array with a character string or a wide-character string. The number of characters in a string must be less than the size of an array to initialize. Continue the compilation neglecting the excess part.</td>
</tr>
<tr>
<td>W1019C</td>
<td>%Z: address of string constant is used for `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] An address of a character string constant is specified as an initial value. Though it does not matter for this system, it may bring a problem for other systems. Continue the compilation making it valid.</td>
</tr>
<tr>
<td>W1020C</td>
<td>%Z: address of static symbol <code>%D' is used for symbol </code>%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The address of a static variable is used as an initial value. It is possible to change the value of the static variable from the outside of the compile unit. Continues the compilation making the initialization valid.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>W1021C</td>
<td>parameter `%D' unused</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A parameter is not referred in the function. Continues the compilation making the parameter valid.</td>
</tr>
<tr>
<td>W1022C</td>
<td>`%D' unused but it is set</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A value is set to a local variable but not referred. Continues the compilation making the variable valid.</td>
</tr>
<tr>
<td>W1023C</td>
<td>`%D' unused</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A local variable is not referred. Continues the compilation making the variable valid.</td>
</tr>
<tr>
<td>W1024C</td>
<td>inefficient operation: result value may be ignored: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation] An operation which has no side effect is specified where a side effect is expected. Continues the compilation leaving it as it is.</td>
</tr>
<tr>
<td>W1025C</td>
<td>statement not reached</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A statement is never reached because of a branch etc. Continues the compilation leaving it as it is.</td>
</tr>
<tr>
<td>W1026C</td>
<td>non void type function `%D' is expected to return value</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A return value must be specified in a return statement of a function unless the function is of the void type. Continues the compilation assuming that an unknown value is returned if no return value is specified.</td>
</tr>
</tbody>
</table>
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1027C</td>
<td>too large long long integer constant %X</td>
</tr>
<tr>
<td></td>
<td><img src="content" alt="Explanation" /></td>
</tr>
<tr>
<td></td>
<td>An integral constant exceeds the maximum value that this system supports. Continues the compilation assuming that the maximum value is specified.</td>
</tr>
<tr>
<td></td>
<td><img src="content" alt="Note" /></td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1028C</td>
<td>empty translation unit</td>
</tr>
<tr>
<td></td>
<td><img src="content" alt="Explanation" /></td>
</tr>
<tr>
<td></td>
<td>A compile unit must contain at least one external declaration. Continues the compilation leaving it as it is.</td>
</tr>
<tr>
<td>W1029C</td>
<td>enumerator list has optional comma</td>
</tr>
<tr>
<td></td>
<td><img src="content" alt="Explanation" /></td>
</tr>
<tr>
<td></td>
<td>No enumerator constant is specified after the last comma of an enumerator constant list. According to the standard, an enumerator constant list does not end with a comma. Continues the compilation neglecting the last comma.</td>
</tr>
<tr>
<td>W1030C</td>
<td>unknown # directive</td>
</tr>
<tr>
<td></td>
<td><img src="content" alt="Explanation" /></td>
</tr>
<tr>
<td></td>
<td>Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system. The indicated directive is unknown for the compiler. Continues the compilation neglecting the directive.</td>
</tr>
<tr>
<td>W1031C</td>
<td>too long identifier, truncated to <code>\%s</code></td>
</tr>
<tr>
<td></td>
<td><img src="content" alt="Explanation" /></td>
</tr>
<tr>
<td></td>
<td>The length of the identifier is larger than the maximum value(65534) the system can manage. Continues the compilation neglecting the excess part.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>W1032C</td>
<td>invalid digit postfix expression</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>W1033C</td>
<td>integer constant out of range</td>
</tr>
<tr>
<td>W1034C</td>
<td><code>;</code> expected at the end of member declaration</td>
</tr>
<tr>
<td>W1035C</td>
<td>character constant too long</td>
</tr>
<tr>
<td>W1036C</td>
<td>newline in string literal</td>
</tr>
</tbody>
</table>
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1037C</td>
<td>numeric octal constant contains <code>8</code> or <code>9</code></td>
<td>The <code>8</code> or <code>9</code> is specified in an octal integer constant. Continues the compilation regarding the <code>8</code> as <code>010</code> and the <code>9</code> as <code>011</code>.</td>
</tr>
<tr>
<td>W1038C</td>
<td>invalid character <code>\\%o</code> (octal)</td>
<td>An invalid character is found in a source program. Continues the compilation neglecting the character.</td>
</tr>
<tr>
<td>W1039C</td>
<td>modulus by 0</td>
<td>A division by zero is performed in a modulus operation. The object for the operation is output. Modify the source code.</td>
</tr>
<tr>
<td>W1040C</td>
<td>division by 0</td>
<td>A division by zero is performed. The object for the operation is output. Modify the source code.</td>
</tr>
<tr>
<td>W1041C</td>
<td>type mismatch in function prototype</td>
<td>Types of parameters do not correspond between function prototype declarations. If void type is specified in a declaration, the same type must be specified in other declarations. Continues the compilation neglecting the declaration.</td>
</tr>
<tr>
<td>W1042C</td>
<td>ellipsis mismatch in function parameter: parameter %d</td>
<td>Positions the ellipsis is specified do not correspond between function prototype declarations. Continues the compilation neglecting the declaration.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>W1043C</td>
<td>type mismatch in function prototype: parameter %d</td>
<td></td>
</tr>
</tbody>
</table>

[Explanation]
Types of parameters do not correspond between function prototype declarations. Continues the compilation neglecting the declaration.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1044C</td>
<td>number of function parameters is different: parameter %d</td>
</tr>
</tbody>
</table>

[Explanation]
Numbers of parameters do not correspond between function prototype declarations. Continues the compilation neglecting the declaration.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1045C</td>
<td>incompatible types: assumed that plain <code>char</code> and <code>%T</code> are compatible types</td>
</tr>
</tbody>
</table>

[Explanation]
The type is incomplete according to the ANSI.

[Note]
This message is not used.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1046C</td>
<td>invalid character <code>\%o</code> (octal) in # directive line</td>
</tr>
</tbody>
</table>

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.

An invalid character is found after the indicated directive. Continues the compilation neglecting the character.

[Note]
This message is not used.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1047C</td>
<td>alert escape sequence is specified</td>
</tr>
</tbody>
</table>

[Explanation]
The `\a` is not the alert escape sequence in the old specification. Continues the compilation regarding the `\a` as the alert escape sequence.
APPENDIX

**W1048C**

promoted to `unsigned int' with unsigned preserved rule

[Explanation]
In the old specification, the integral promotion is made according to the unsigned preserving rule. Continues the compilation assuming that the value is promoted to the unsigned int type.

**W1049C**

promoted to `int' with value preserved rule

[Explanation]
In the ANSI, the integral promotion is made according to the unsigned preserving rule. Continues the compilation assuming that the value is promoted to the int type.

**W1050C**

cannot concatenate character string literal and wide character string literal: assumed as character string literal

[Explanation]
Cannot concatenate a character string literal and a wide character string literal. It is regarded as a character string literal.

[Note]
This message is not used.

**W1051C**

cannot concatenate wide character string literal and character string literal: assumed as wide character string literal

[Explanation]
Cannot concatenate a character string literal and a wide character string literal. It is regarded as a wide character string literal.

**W1052C**

length of identifier `%s' is longer than %d

[Explanation]
An identifier is too long. Continues the compilation making the whole length valid.

[Note]
This message is not used.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1053C</td>
<td>incompatible types: assumed that pointer to char and</td>
<td>Types of a pointer to char and a pointer to void are regarded to be compatible. Continues</td>
</tr>
<tr>
<td></td>
<td>pointer to void are compatible types</td>
<td>the compilation assuming that they are compatible types.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1054C</td>
<td>incompatible types between pointer to void and pointer</td>
<td>According to the ANSI types of a pointer to void and a pointer to function are regarded to</td>
</tr>
<tr>
<td></td>
<td>to function</td>
<td>be incompatible. Continues the compilation assuming that they are compatible types.</td>
</tr>
<tr>
<td>W1055C</td>
<td>incompatible types: assumed that pointer to void and</td>
<td>Types of a pointer to void and a pointer to non-void are regarded to be compatible. Continues</td>
</tr>
<tr>
<td></td>
<td>pointer to non-void are compatible types</td>
<td>the compilation assuming that they are compatible types.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1056C</td>
<td>invalid field `%I' is used</td>
<td>Though the indicated member name is not found in any struct nor union, it is found as a plain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>symbol. Continues the compilation assuming that it is a plain symbol according to the old</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specification.</td>
</tr>
<tr>
<td>W1057C</td>
<td>cast to incomplete type `%T'</td>
<td>An enumeration type whose tag is not defined is specified as the target type of a type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>conversion. Continues the compilation assuming that it is converted to the int type.</td>
</tr>
</tbody>
</table>
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1058C</td>
<td>identifier <code>%D</code> is used out of its scope</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>An identifier which is declared as an external</td>
</tr>
<tr>
<td></td>
<td>reference symbol in a block is referred outside</td>
</tr>
<tr>
<td></td>
<td>the block. Continues the compilation making the</td>
</tr>
<tr>
<td></td>
<td>reference valid.</td>
</tr>
<tr>
<td>W1059C</td>
<td>This behavior is ANSI C undefined behavior</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The operation is regarded as undefined in the</td>
</tr>
<tr>
<td></td>
<td>ANSI C.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1060C</td>
<td>length of external linkage identifier (%D %d) is</td>
</tr>
<tr>
<td></td>
<td>longer than %d</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The length of an identifier for the external</td>
</tr>
<tr>
<td></td>
<td>linkage is too long.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1061C</td>
<td>length of internal linkage identifier (%D %d) is</td>
</tr>
<tr>
<td></td>
<td>longer than %d</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The length of an identifier for the internal</td>
</tr>
<tr>
<td></td>
<td>linkage is too long.</td>
</tr>
<tr>
<td>W1062C</td>
<td>void type parameter name <code>%D</code> is ignored</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>If the type of a parameter for a function is</td>
</tr>
<tr>
<td></td>
<td>void, no identifier is needed for a parameter.</td>
</tr>
<tr>
<td></td>
<td>Continue the compilation neglecting the</td>
</tr>
<tr>
<td></td>
<td>identifier.</td>
</tr>
</tbody>
</table>
W1063C  parameter type have to be compatible with the type that results from the application of default type promotions

[Explanation]
Types of parameters for a prototyped function and a non-prototyped function are compatible if the resulting type of the default argument promotion made on the prototype matches with the original type for each parameter.
Continue the compilation making the types in the prototype valid.

W1064C  prototype declaration of `\%D' but previous traditional definition

[Explanation]
Though a non-prototype function definition is already made, a prototype function declaration is also made. Continue the compilation making the declaration valid.

W1065C  inefficient identifier without declaration specifier in function parameter

[Explanation]
An identifier without a declaration specifier is specified as a parameter in a function declaration. Continue the compilation neglecting the parameter.

W1066C  static function `\%D' is used but not defined

[Explanation]
Though a static function is declared and referred, no definition is found. Continue the compilation leaving it as it is.

W1067C  static function `\%D' is declared but not defined

[Explanation]
Though a static function is declared, no definition and reference is found. The declaration is meaningless. Continue the compilation leaving it as it is.
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1068C</td>
<td>Case value is out of range: type of case expression is inconsistent with type of switch conditional expression: case value is truncated</td>
</tr>
</tbody>
</table>

**Explanation**

The value specified in a case label cannot be expressed by the type of the condition expression for the switch statement. The value in the case label is rounded down to match with the type.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1069C</td>
<td>Decimal integer constant <code>%s</code> is too large</td>
</tr>
</tbody>
</table>

**Explanation**

The decimal integer constant is too large to express internally. Continue the compilation making lower bits valid.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1070C</td>
<td>Octal integer constant <code>%%s</code> is too large</td>
</tr>
</tbody>
</table>

**Explanation**

The octal integer constant is too large to express internally. Continues the compilation making lower bits valid.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1071C</td>
<td>Hexadecimal integer constant <code>0x%s</code> is too large</td>
</tr>
</tbody>
</table>

**Explanation**

The hexadecimal integer constant is too large to express internally. Continues the compilation making lower bits valid.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1072C</td>
<td>Binary integer constant <code>%s</code> is too large</td>
</tr>
</tbody>
</table>

**Explanation**

The binary integer constant is too large to express internally. Continues the compilation making lower bits valid.

**Note**

This message is not used.
Appendix E  ERROR MESSAGE

W1073C  decimal integer constant is too large

[Explanation]
The decimal integer constant specified as a line number is too large to express internally. Continues the compilation making lower bits valid.

[Note]
This message is not used.

W1074C  floating point constant `%s' is out of range of `float'

[Explanation]
A floating point constant is out of range for the float type. Continues the compilation converting it to the double type and rounding it off.

W1075C  floating point constant `%s' is out of range of `double'

[Explanation]
A floating point constant is out of range for the double type. If it is overflowed, the resulting value is +HUGE_VAL or -HUGE_VAL. If it is underflowed, the resulting value is 0.

W1076C  floating point constant `%s' is out of range of `long double'

[Explanation]
A floating point constant is out of range for the long double type.

[Note]
This message is not used.

W1077C  external declaration has no declaration specifier

[Explanation]
No declaration specifier is specified in an external variable declaration. Only for a function definition a declaration specifier is omitted in an external declaration. Continues the compilation assuming that "int" is specified.
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1078C</td>
<td><code>%s</code> is ANSI C extension</td>
<td>The &quot;__asm&quot; is specification of the ANSI C extension. Continues the compilation leaving it as it is.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1079C</td>
<td>prototype parameter declaration is used</td>
<td>A function prototype is used. Continues the compilation leaving it as it is.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1080C</td>
<td><code>%s</code> is used</td>
<td>An asm statement is found. Continues the compilation leaving it as it is.</td>
<td></td>
</tr>
<tr>
<td>W1081C</td>
<td>multiple type qualifier specified in declaration with obsolete modified typedef type</td>
<td>Though a type qualifier is specified in a typedef declaration, it is specified together with the typedef name. The effect of the qualifier is the same as one qualifier is specified.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1082C</td>
<td>multiple <code>const</code> specified</td>
<td>Several &quot;const&quot;, which effects on a symbol or pointer, are specified. The effect of all of them is equal to that of one &quot;const&quot;.</td>
<td></td>
</tr>
</tbody>
</table>
Appendix E ERROR MESSAGE

W1083C multiple `volatile’ specified

[Explanation]
Several "volatile", which effects on a symbol or pointer, are specified. The effect of all of them is equal to that of one "volatile".

W1084C typedefed type already qualified with `const’

[Explanation]
Though the "const" is specified in a typedef declaration, it is specified together with the typedef name. The effect of the "const" is the same as one "const" is specified.

W1085C typedefed type already qualified with `volatile’

[Explanation]
Though the "volatile" is specified in a typedef declaration, it is specified together with the typedef name. The effect of the "volatile" is the same as one "volatile" is specified.

W1086C duplicate storage class specifier specified for declaration of `%s’

[Explanation]
The same storage class specifiers appear in a declaration. Continues the compilation making one specifier valid.

W1087C no name specified in declaration

[Explanation]
No identifier is specified in a declaration. An identifier is omitted only on a tag declaration. Continues the compilation neglecting the declaration.

W1088C type declaration `%T’ without body in block hides previous type

[Explanation]
A struct or union whose tag is not declared is specified in a block. Though it is usually specified to make the previous declaration invisible, assure that it is used as is intended including back references. Continues the compilation making the declaration valid.
## APPENDIX

<table>
<thead>
<tr>
<th>W1089C</th>
<th>W1090C</th>
<th>W1091C</th>
<th>W1092C</th>
<th>W1093C</th>
<th>W1094C</th>
</tr>
</thead>
<tbody>
<tr>
<td>lacked tag name in struct/union declaration</td>
<td>tag is declared first in parameter list: `%E %I' has function prototype scope</td>
<td>redeclaration of `%D'</td>
<td>redeclaration of `%D': linkage conflict</td>
<td>redeclaration of builtin symbol `%D' as static function</td>
<td>redeclaration of `%D': promoted parameter mismatch</td>
</tr>
</tbody>
</table>

**[Explanation]**

A tag name must be specified in a declaration of a struct or union tag. A tag cannot be used with a member declaration only. Continues the compilation leaving it as it is.

**[Explanation]**

A tag appears for the first time in a parameter of a function prototype declaration. The tag is visible only within the prototype declaration, then the type of an argument may be incompatible in a function call.

Continues the compilation making the tag declaration valid.

**[Explanation]**

A symbol is redeclared. Assure that the type is compatible.

**[Explanation]**

The declaration of the symbol is regarded as a redeclaration because the linkage conflicts. Check a storage class specifier.

**[Explanation]**

Though a built-in function, which is defined by the system, is regarded as a function of the external linkage, it is redeclared as a function of the static storage class.

**[Explanation]**

The declaration is regarded as a redeclaration because the resulting parameter type of the default argument promotion conflicts. Assure that the type specifier is correct.
Appendix E  ERROR MESSAGE

W1095C  linkage conflict between internal and external: `D'

[Explanation]
The linkage of the symbol conflicts. Assure that the storage class specifier is correct.

W1096C  %s: definition of `s'

[Explanation]
In a function definition, the linkage conflicts with that previously declared symbol. Check the storage class specifier.

W1097C  %s: declaration of `s'

[Explanation]
In a function declaration, the linkage conflicts with that previously declared symbol. Check the storage class specifier.

W1098C  %s: declaration of `s'

[Explanation]
In a function declaration, the linkage conflicts with that previously declared symbol. Check the storage class specifier.

W1099C  external array `D' has variable dimension

[Explanation]
The subscript of an array which is declared externally is not an integral constant.

[Note]
This message is not used.

W1100C  aggregate type symbol `D' declared with register storage class

[Explanation]
A symbol of the array, struct or union type was declared with the "register" storage class specifier. It is valid, but not portable.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1101C</td>
<td>local declaration of `%D' hides parameter</td>
<td>The symbol which is already declared as a parameter is declared as a local declaration. It is a redeclaration according the standard.</td>
</tr>
<tr>
<td>W1102C</td>
<td>declaration of `%D' hides previous declaration</td>
<td>The declaration of a symbol in a block makes the same symbol declared outside the block invisible. The symbol declared outside is invisible.</td>
</tr>
<tr>
<td>W1103C</td>
<td>subscript of array is zero</td>
<td>The value of a subscript of an array is equal to zero. Continues the compilation making the declaration valid.</td>
</tr>
<tr>
<td>W1104C</td>
<td>useless type qualifiers for function return type: `%T'</td>
<td>A type qualifier (const, volatile) specified to a return value of a function is meaningless.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1105C</td>
<td>useless type qualifier specified</td>
<td>A type qualifier is specified to a tag declaration. A type qualifier is effective for a symbol or a pointer and meaningless here.</td>
</tr>
<tr>
<td>W1106C</td>
<td>function return block scope type: `%T'</td>
<td>The type of the return value for the function is the struct or union declared inside the block. It may bring an incompatibility with the type of the function.</td>
</tr>
</tbody>
</table>
Appendix E  ERROR MESSAGE

W1107C  escape sequence `\x' is specified

[Explanation]
The representation using `\x' of a hexadecimal is invalid in the old specification.

W1108C  useless type qualifiers for array type

[Explanation]
A type qualifier specified to an array is effective for its element.

[Note]
This message is not used.

W1109C  useless type qualifiers for function type

[Explanation]
A type qualifier (const, volatile) specified to a return value of a function is meaningless.

W1110C  parameter `%D' uses block scope struct/union type: `%T'

[Explanation]
The type of a parameter for a function is a struct or union type declared inside the block. It may bring an incompatibility with the type of an argument.

W1111C  void type parameter name `%D' is ignored

[Explanation]
No parameter name is needed for a void type parameter. The parameter type of void means that a function takes no parameter.

[Note]
This message is not used.
APPENDIX

W1112C  useless storage class specifier `%A' specified

[Explanation]
A storage class specifier is specified in a tag declaration. It specifies a feature of a symbol, so meaningless here.

W1113C  type qualifiers of parameter void type is ignored

[Explanation]
A type qualifier is meaningless for a void parameter. The parameter type of void means that a function takes no parameter.

[Note]
This message is not used.

W1114C  parameter identifier `%I' is implicitly declared as `int'

[Explanation]
In an old style definition of a function, the type of a symbol which is specified as a parameter is not specified. Assume that it is parameter of the int type.

W1115C  type declared in cast expression

[Explanation]
A tag of a struct or union is declared in a cast expression.

W1116C  external declaration `%D' with initializer at %s

[Explanation]
An external variable with the "extern" storage class specifier is initialized. The "extern" storage class specifier represents that a variable is external reference and it is meaningless.
W1117C  type of bit-field member `%D' is not `int', `signed int' or `unsigned int'

[Explanation]
According to the standard, the type of a bit-field is `int', `signed int' or `unsigned int'. Other types are recognized as the extension.

W1118C  bit-field needs signed or unsigned: type of bit-field `%D' is assumed `%T'

[Explanation]
The sign of a bit-field with neither signed nor unsigned specified depends on systems. The resulting type of an internal conversion is applied.

W1119C  enumerator value exceeds INT_MAX

[Explanation]
The maximum value for the enumeration is the value represented by INT_MAX. Continues the compilation assuming that INT_MIN is specified.

W1120C  type of bit-field is `%T': type of bit-field `%D' is assumed `%T'

[Explanation]
The enum type is specified for a bit-field. The resulting type of an internal conversion is applied.

W1121C  invalid operands: %s: %O

[Explanation]
A pointer to the function is specified in the pointer type operation (addition, subtraction, comparison). These operators require a pointer to an object for a pointer type.

W1122C  %Z: `%T' has `const' field: %O

[Explanation]
Cannot substitute to a struct whose member is qualified as const.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1123C</td>
<td>%Z incompatible pointer type in function `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Types of pointers are incompatible.</td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>W1124C</td>
<td>%Z: %s: `%T': %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A type whose size cannot be determined (array without subscript, function or void) is specified. The size of an array without subscript is assumed to be 0, that of the function and void type 1.</td>
</tr>
<tr>
<td>W1125C</td>
<td>%Z: %s: `%T'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A type whose size cannot be determined (array without subscript, function or void) is specified. The size of an array without subscript is assumed to be 0, that of the function and void type 1.</td>
</tr>
<tr>
<td>W1126C</td>
<td>%Z incompatible pointer type: `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Types of pointers are incompatible.</td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>W1127C</td>
<td>type conversion void to void</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A cast operation is made from the void type to the void type.</td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message Content</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>W1128C</td>
<td>invalid type conversion to void type</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A cast operation is made to the void type.</td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>W1129C</td>
<td>invalid type conversion to same non scalar type</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The type of a destination for a cast operation and that of an operand are the same struct or union. The cast operation is useless.</td>
</tr>
<tr>
<td>W1130C</td>
<td>invalid type for conditional expression: `%T' specified: scalar type expected: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The void type is specified to the first operand of the conditional operator.</td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
</tr>
<tr>
<td>W1131C</td>
<td>`%D' may be used before it is set</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A symbol is referred before its value is set. The resulting value is not safe.</td>
</tr>
<tr>
<td>W1132C</td>
<td>implicitly function declared <code>%I': assumed return </code>int'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A function is called before declared. The type of the return value is assumed as the `int' type.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>W1133C</td>
<td>return without value in non void type function `%D'</td>
</tr>
</tbody>
</table>

[Explanation]
No return value is specified in a return statement in a function which is not of the void type.

| W1134C | unmodifiable object `%D'                                               |

[Explanation]
Cannot change the value of the const-qualified symbol.

| W1135C | unmodifiable address                                                  |

[Explanation]
Cannot change the value of the const-qualified pointer.

| W1136C | implicitly take address of rvalue                                     |

[Explanation]
The address of rvalue is used implicitly.

[Note]
This message is not used.

| W1137C | take address of rvalue                                               |

[Explanation]
An object to take an address must be lvalue.

| W1138C | cannot take address of cast expression                              |

[Explanation]
An object to take an address must be lvalue. The result of a cast operation is not a lvalue.
W1139C cannot take address of `%D' declared with `register' storage class specifier

[Explanation]
The address of a symbol is not available if the symbol is declared with the "register" storage class specifier.

W1140C parameter `%D' cannot be redeclared in function body

[Explanation]
The symbol which is already declared as a parameter is declared as a local symbol. A parameter cannot be redeclared.

W1141C incompatible types between `%T' and `%T'

[Explanation]
An information of types causing a type incompatibility.

W1142C redefinition of %s `%D'

[Explanation]
The typedef name cannot be redefined. Because it does not have linkage, the typedef cannot be specified two or more time.

W1143C %Z incompatible pointer type: argument %d of `%D'

[Explanation]
An incompatible pointer.

[Note]
This message is not used.

W1144C %Z: expected `%T' actual `%T': argument %d of `%D'

[Explanation]
Types of an argument for function call and of a parameter at declaration are incompatible. Modify the argument to match with the parameter.
APPENDIX

W1145C %O applys to bit-field

[Explanation]
A bit-field is specified as an operand of the sizeof operator. The address of a bit-field is not available.

W1146C `$' character in identifier

[Explanation]
The `$' is used in an identifier.

[Note]
This message is not used.

W1147C take address of rvalue: %O

[Explanation]
The address of a rvalue (the right-hand side value of an assignment operation) is taken.

W1148C invalid `long float' type specifier: treated as `double'

[Explanation]
The long float type is specified. It is an invalid type according to the standard. Regard it as the double type.

W1149C fixed parameter required before `...' .

[Explanation]
It is invalid to specify an ellipsis alone in a parameter list. A parameter is needed beside it.

W1150C medium type specifier is used

[Explanation]
The medium type specifier is invalid.

[Note]
This message is not used.
Appendix E  ERROR MESSAGE

W1151C  %Z: expected `%T' actual `%T' in function `%D'

[Explanation]
The type of the return value for a function and that of an expression in a return statement are incompatible. Modify the expression in the return statement to match with the type of the return value.

W1152C  %Z: expected `%T' actual `%T': argument %d

[Explanation]
Types of an argument for function call and of a parameter at declaration are incompatible. Modify the argument to match with the parameter.

W1153C  %Z from `%T' to `%T': `%D'

[Explanation]
Types of an initializer and of the target are incompatible. Modify the initializer to match with the target type.

W1154C  %Z from `%T' to `%T': %O

[Explanation]
Types of both sides for the assignment operation are incompatible. Modify the right-hand side value to match with the type of the left-hand side.

W1155C  %Z incompatible pointer type: argument %d

[Explanation]
An incompatible pointer.

[Note]
This message is not used.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1156C</td>
<td>%Z between incompatible pointer types: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>An incompatible pointer.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1157C</td>
<td>non void type function `%D' reaches to the end of function without return</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>No return statement is written in a function of a non-void type. An unknown value is returned.</td>
</tr>
<tr>
<td>W1158C</td>
<td>function <code>%D' returns </code>int'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>No declaration specifier is written in a function definition. Assume that the function returns an int type value, for no type information is available.</td>
</tr>
<tr>
<td>W1159C</td>
<td>linkage conflict between internal and external</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The linkage does not match with that defined symbol previously.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is sub message of W1096C,W1097C,W1098C.</td>
</tr>
<tr>
<td>W1160C</td>
<td>%Z: left hand side must be modifiable lvalue</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The left-hand side must be modifiable lvalue.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>W1161C</td>
<td>reference to rvalue array</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>An array as rvalue is referred.</td>
</tr>
<tr>
<td>W1162C</td>
<td>storage class specifier <code>%A</code> is specified for function <code>%D</code> declared in block</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>A function is declared with the &quot;static&quot; storage class specifier in a block. According to the standard, an explicit storage class specifier except for the &quot;extern&quot; cannot be specified in a function declaration in a block.</td>
</tr>
<tr>
<td>W1163C</td>
<td>extern function <code>%D</code> declared in block</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>A function is declared with the &quot;extern&quot; storage class specifier in a block.</td>
</tr>
<tr>
<td>W1164C</td>
<td>'long long' integer constant is used</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The integral constant of the long long type which two postfixes 'L' or 'l' is specified is specification of the ANSI extension. However the long long type is not supported in this system.</td>
</tr>
<tr>
<td>W1165C</td>
<td>parameter type mismatch of <code>%D</code></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>When a function is declared without a parameter information a default argument promotion is made on the prototype. The resulting type of the default argument promotion is incompatible with the original type. Modify the function prototype declaration to match with the resulting type.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>W1166C</td>
<td>value of integral constant expression for enumerator <code>%I</code> is out of range</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The number set to an enumerator constant exceeds the INT_MAX. The number must be expressible by the <code>int</code> type.</td>
</tr>
<tr>
<td>W1167C</td>
<td>unary minus operator applies to too large unsigned integer constant</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The unary <code>-</code> operator is applied to an integral constant of the unsigned type.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1168C</td>
<td><code>%E %I</code> declared in parameter declaration</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>A tag is declared in a parameter declaration of a non-prototype function definition at the same time. The scope where the tag is visible is the block scope from the tag declaration to the end of the function, then a parameter type mismatch may occur in a function call. Continues the compilation making the declaration valid.</td>
</tr>
<tr>
<td>W1169C</td>
<td>label <code>%D</code> is not referred</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The label is not referred by any goto statement. It does not bring a problem to a program.</td>
</tr>
<tr>
<td>W1170C</td>
<td>`long long' type specifier is used</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The long long type is specified for type specifier. The type is not supported in this system. Continues the compilation regarding the type as the long type.</td>
</tr>
</tbody>
</table>
### Appendix E  ERROR MESSAGE

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1171C</td>
<td>unterminated filename</td>
<td>Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system. One of those directives is the line directive, which represents a line number and a file name. The double quotation to terminate the file name is not written. Continues the compilation neglecting the directive.</td>
</tr>
<tr>
<td>W1172C</td>
<td>too long hexadecimal escape sequence</td>
<td>The hexadecimal by the escape sequence <code>\x</code> cannot be expressed by a character. Continues the compilation making only the lowest byte or wide-character of the resulting value valid.</td>
</tr>
<tr>
<td>W1173C</td>
<td><code>*/</code> exists outside of comment</td>
<td>Though a comment must be enclosed by the <code>/*</code> and <code>*/</code> in the C language, <code>*/</code> is found outside the comment. A comment cannot be nested. Continues the compilation regarding those simply as the <code>*</code> and the <code>/</code>.</td>
</tr>
<tr>
<td>W1174C</td>
<td>hexadecimal escape sequence has no digit value</td>
<td>No hexadecimal digit is written after the <code>\x</code> escape sequence. Continues the compilation assuming that a <code>\x0</code> is written.</td>
</tr>
<tr>
<td>W1175C</td>
<td>type conversion between different integral types</td>
<td>A type conversion is made between integral types with different sizes.</td>
</tr>
</tbody>
</table>

[Note]
This message is not used.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1176C</td>
<td>type conversion between different floating-points</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td>A type conversion is made between floating point types with different sizes.</td>
</tr>
<tr>
<td>W1177C</td>
<td>truncate pointer to lose significance: `%s%s'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td>A pointer type is converted to an integral type which is not capable for the address value the pointer has. The value is meaningless as an address value.</td>
</tr>
<tr>
<td>W1178C</td>
<td>integral type expression expected for <code>\%%%c: function </code>%s' argument %d'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td>A pointer (character string) to the `char' type cannot be specified for conversion specifiers as %d, %u, &amp;o and %i in a format character string of the printf.</td>
</tr>
<tr>
<td>W1179C</td>
<td>integral type expression expected for <code>\%%%c: function </code>%s' argument %d'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td>A pointer cannot be specified for the conversion specifier %c in a format character string of the printf.</td>
</tr>
<tr>
<td>W1180C</td>
<td>pointer to void type expression expected for <code>\%%%c: function </code>%s' argument %d'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td>Only the pointer to the void type can be specified for the conversion specifier %p in a format character string of the printf.</td>
</tr>
<tr>
<td>W1181C</td>
<td>pointer to integral type expression expected for <code>\%%%c: function </code>%s' argument %d'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td>Only the pointer to the integral type can be specified for the conversion specifier %n in a format character string of the printf.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>W1182C</td>
<td>too many arguments for format string: function `%s'</td>
<td></td>
</tr>
</tbody>
</table>
|            | [Explanation]  
|            | Too many arguments are specified for the conversion specifier in a format character string of the printf. |
| W1183C     | invalid struct/union member declaration: name required |
|            | [Explanation]  
|            | No declarator is specified for a declaration of the member for the struct or union. Specify either an empty bit-field (:0 is specified) or a member name. |
| W1184C     | expand pointer to lose significance: `\%s\%s' |
|            | [Explanation]  
|            | A pointer type is converted to an integral type which is larger than the size of the address value the pointer has. The value is meaningless as an address value. |
| W1185C     | redeclaration of global symbol `\%D' as extern symbol declared in block |
|            | [Explanation]  
|            | The type of a symbol declared in a block with the "extern" storage class specifier is incompatible with that of the external symbol declared previously or external reference declared. Modify them to match the type. |
| W1186C     | unknown escape sequence `\\\%c' |
|            | [Explanation]  
|            | A character which cannot be regarded as an escape sequence in the ANSI C is found after the `\'. Continues the compilation neglecting the `\'. |
| W1187C     | too large integer constant for radix %d |
|            | [Explanation]  
<p>|            | The digit-sequence is too long and surely overflows before converting it into the internal format. Continues the compilation assuming that 0 is specified. |</p>
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| W1188C  | escape sequence does not fit in range of character | [Explanation]  
The result of the escape sequence cannot be represented by a byte (ex. \\
|         |                                              | "\400"). Continues the compilation making only the lowest byte valid.                                                                                                                                                                                                                                                                                                                                                                  |
| W1189C  | escape sequence does not fit in range of wide character | [Explanation]  
The result of the escape sequence cannot be represented by two bytes. Continues the compilation making only the lowest word valid.                                                                                                                                                                                                                                                                                                                                 |

[Note]  
This message is not used.

| W1190C  | too large integer constant `\%s` for radix \%d | [Explanation]  
The digit-sequence is too long and surely overflows before converting it into the internal format. Continues the compilation assuming that 0 is specified.                                                                                                                                                                                                                                                                 |

| W1191C  | arithmetic calculation/conversion error: \%O | [Explanation]  
A constant operation results in an error.                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

| W1192C  | exceed the maximum length of octal escape sequence | [Explanation]  
More than four figures are specified in an octal escape sequence.                                                                                                                                                                                                                                                                                                                                                                                                                           |

[Note]  
This message is not used.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1193C</td>
<td>different symbol types between <code>%s' and </code>%s'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The same identifier is used in different ways in the same name space.</td>
</tr>
<tr>
<td>W1194C</td>
<td>unknown format flags 0x%x: function `%s'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] An unknown conversion specifier is found in a format character string of the printf.</td>
</tr>
<tr>
<td>W1195C</td>
<td>unknown format flags %%%%c: function `%s'</td>
</tr>
<tr>
<td></td>
<td>[Explanation] An unknown conversion specifier is found in a format character string of the printf.</td>
</tr>
<tr>
<td>W1196C</td>
<td>function return type `%T' is not promoted for function type declaration without prototype</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Though the default argument promotion is made for the type of the return value for the function in the old standard, the type remains as is specified in the ANSI.</td>
</tr>
<tr>
<td>W1197C</td>
<td>integer constant isn't treated as unsigned: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation] Regard the unsigned integral constant as a signed one.</td>
</tr>
<tr>
<td>W1198C</td>
<td>`%T' is useless declaration</td>
</tr>
<tr>
<td></td>
<td>[Explanation] It is meaningless to declare an enumeration tag (enum tag) alone. The tag is ignored.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>W1199C</td>
<td>array of incomplete type <code>%T</code></td>
</tr>
<tr>
<td>W1200C</td>
<td>suspicious assignment operator in conditional expression: <code>%O</code></td>
</tr>
<tr>
<td>W1201C</td>
<td>different semantics between transition and later: <code>%O</code></td>
</tr>
<tr>
<td>W1202C</td>
<td>call function without prototype declaration: <code>%s</code></td>
</tr>
<tr>
<td>W1203C</td>
<td>use address of string constant in conditional expression</td>
</tr>
<tr>
<td>W1204C</td>
<td>static symbol <code>%D</code> unused</td>
</tr>
</tbody>
</table>
Appendix E  ERROR MESSAGE

W1205C  %O: constant out of range for `%T': %X

[Explanation]
A constant specified in a comparative operation or in a case label exceeds the range of the type. Modify the value of the constant.

W1206C  string expression expected for %%%c: function `%%s' argument %d

[Explanation]
Only a pointer (e.g. character string) to a character string can be specified for the conversion specifier %s in a format character string of the printf.

W1207C  conditional expression is constant: %O

[Explanation]
The expression of a condition is constant. Assure that it is as is intended.

W1208C  cannot take address of array `%%D' declared with `register'

[Explanation]
Attempted to take an address of an array. The array is declared with the "register" storage class specifier and it is not possible to get the address of it. Remove the "register" storage class specifier to take an address.

W1209C  too many format characters specified in string constant for function `%%s'

[Explanation]
Too few arguments are specified for conversion specifiers in a format character string of the printf. No check is done for conversion specifiers for which no arguments are specified.

W1210C  %Z: constant out of range for `%T': %X

[Explanation]
A constant specified as an initializer exceeds the range of the type. Modify the value of the constant.
APPENDIX

W1211C  long double expression expected for %L%c: function `s' argument %d

[Explanation]
Only the long double type can be specified for conversion specifiers as %Lf, %Le, %LE, %Lg and %LG in a format character string of the printf.

W1212C  floating-point type expression expected for %c: function `s' argument %d

[Explanation]
Only the floating point type (float, double and long double) can be specified for conversion specifiers as %f, %e, %E, %g and %G in a format character string of the printf.

W1213C  constant out of range for `T' bit-field: its width %d: %O

[Explanation]
A constant specified in a comparative operation exceeds the range of a bit-field. Modify the value of the constant.

W1214C  type conversion between different types: `T' and `T'

[Explanation]
Attempted to convert a null pointer constant to a type other than the pointer type.

W1215C  end of loop not reached

[Explanation]
The end part of a loop is never reached because a statement to break the loop (break, goto) exists in the loop.

W1216C  loop not entered from the entry of loop

[Explanation]
The goto statement and the label jumps over the head part of a loop. The part is not executed.
W1217C | old-fashioned function declaration hides previous prototype declaration `%D'

[Explanation]
A non-prototype function declaration makes a prototype one invisible. Parameter check is not performed in this case, so prefer a prototype declaration if possible.

W1218C | `%D' hides external

[Explanation]
A symbol declaration makes an external symbol invisible.

[Note]
This message is not used.

W1219C | `const' symbol `%D' has no initializer

[Explanation]
A variable declared with the "const" qualifier is not initialized. The value of such a variable cannot be changed, so, in this case, it is used with its value unknown.

W1220C | static `%D' hides external

[Explanation]
A symbol declared with the "static" storage class specifier makes an external symbol invisible. The external variable is invisible.

W1221C | typedef `%D' hides external

[Explanation]
A symbol declared with the "typedef" storage class specifier makes an external symbol invisible. The external variable is invisible.

W1222C | auto/register `%D' hides external

[Explanation]
A symbol declaration with the "auto" or "register" storage class specifier makes an external symbol invisible. The external variable is invisible.
W1223C invalid function type conversion from `\%T\' to `\%T\'

[Explanation]
Pointers to the function type and to a non-void type cannot be converted to each other.

W1224C incompatible enumeration between `\%T\' and `\%T\'

[Explanation]
Types are incompatible.

[Note]
This message is not used.

W1225C type of expression is incomplete type: `\%T\'

[Explanation]
The type of an expression is a struct or union type whose tag is not defined. A struct or union whose tag is not defined cannot be specified in an expression. Define the tag.

W1226C type of expression is `\%T\'

[Explanation]
An object is referred by a pointer to the void type. A void type object cannot be referred. Cast it to a proper type before refer it.

W1227C arithmetic calculation overflow: %O

[Explanation]
An arithmetic operation results in an overflow. Inspect a constant operation.

W1228C arithmetic calculation underflow: %O

[Explanation]
An arithmetic operation results in an underflow. Inspect a constant operation.
Appendix E ERROR MESSAGE

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1229C</td>
<td>floating point exception: %O</td>
<td>An arithmetic operation results in a floating point exception. Inspect a constant operation.</td>
</tr>
<tr>
<td>W1230C</td>
<td>floating point division by 0: %O</td>
<td>A floating point division by zero is performed in an arithmetic operation. Inspect a constant operation.</td>
</tr>
<tr>
<td>W1231C</td>
<td>floating point overflow: %O</td>
<td>In an arithmetic operation, a floating point overflow occurred. Inspect a constant operation.</td>
</tr>
<tr>
<td>W1232C</td>
<td>floating point underflow: %O</td>
<td>In an arithmetic operation, a floating point underflow occurred. Inspect a constant operation.</td>
</tr>
<tr>
<td>W1233C</td>
<td>floating point inexact: %O</td>
<td>In an arithmetic operation, the result of a floating point operation is not exact. Inspect a constant operation.</td>
</tr>
<tr>
<td>W1234C</td>
<td>conversion to <code>\%T</code> is out of range</td>
<td>In a type conversion of a constant, the resulting value of a type conversion for a constant is out of range. Inspect a constant operation.</td>
</tr>
<tr>
<td>W1235C</td>
<td>arithmetic calculation is out of range: %O</td>
<td>In an arithmetic operation, the resulting value of an operation is out of range. Inspect a constant operation.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>W1236C</td>
<td>invalid type combination of <code>%T</code> and <code>%T</code>: %O</td>
<td></td>
</tr>
<tr>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The combination for types of operands is invalid.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1237C</td>
<td>invalid type combination of <code>%T</code> and <code>%T</code>: integral type required: %O</td>
<td></td>
</tr>
<tr>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The combination for types of operands is invalid.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1238C</td>
<td>invalid type combination of <code>%T</code> and <code>%T</code>: arithmetic type required: %O</td>
<td></td>
</tr>
<tr>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The combination for types of operands is invalid.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1239C</td>
<td>invalid type combination of <code>%T</code> and <code>%T</code>: void type required: %O</td>
<td></td>
</tr>
<tr>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The combination for types of operands is invalid.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1240C</td>
<td>invalid type combination of <code>%T</code> and <code>%T</code>: compatible types required: %O</td>
<td></td>
</tr>
<tr>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The combination for types of operands is invalid.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1241C</td>
<td>unknown escape sequence <code>\%o</code></td>
<td></td>
</tr>
<tr>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A character which is not specified as an escape sequence in the ANSI C is found after the <code>\</code>. Continues the compilation neglecting the <code>\</code>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix E  ERROR MESSAGE

W1242C  return address of local variable

[Explanation]
A local variable is allocated in a stack area. If a function called returns an address of a local variable declared in it and the variable is referenced by the pointer, it is not sure that the value is safe.

W1243C  return address of parameter

[Explanation]
A parameter is allocated in a stack area. If a function called returns an address of a parameter declared in it and the variable is referenced by the pointer, it is not sure that the value is safe.

W1244C  builtin function `%D' is redefined

[Explanation]
The built-in function is defined by the system internally and types of the return value and parameters are already defined. Do not redefine the function.

W1245C  multiple `__interrupt' specified

[Explanation]
Several "__interrupt", which effects on a symbol or pointer, are specified. The effect for all of them is equal to that of one "__interrupt".

W1246C  multiple `__subinterrupt' specified

[Explanation]
Several "__subinterrupt", which effects on a symbol or pointer, are specified. The effect for all of them is equal to that of one "__subinterrupt".

W1247C  multiple `__io' specified

[Explanation]
Several "__io", which effects on a symbol or pointer, are specified. The effect for all of them is equal to that of one "__io".
APPENDIX

W1248C typedefed type already qualified with `__interrupt`

[Explanation]
Though the "__interrupt" is specified in a typedef declaration, it is specified together with the typedef name. The effect of the "__interrupt" is the same as one "__interrupt" is specified.

W1249C typedefed type already qualified with `__subinterrupt`

[Explanation]
Though the "__subinterrupt" specified in a typedef declaration, it is specified together with the typedef name. The effect of the "__subinterrupt" is the same as one "__subinterrupt" is specified.

W1250C typedefed type already qualified with `__io`

[Explanation]
Though the "__io" is specified in a typedef declaration, it is specified together with the typedef name. The effect of the "__io" is the same as one "__io" is specified.

W1251C __interrupt or __subinterrupt do not operate on function declarator

[Explanation]
The "__interrupt" and the "__subinterrupt" are type qualifiers effective only on a function declarator. Assure that a valid type qualifier is specified. Continues the compilation neglecting the "__interrupt" or the "__subinterrupt".

W1252C subinterrupt function does not called interrupt function

[Explanation]
A __subinterrupt function is not called by an interrupted function.
### Appendix E  ERROR MESSAGE

<table>
<thead>
<tr>
<th>W1253C</th>
<th>__io do not operate on external variable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__io&quot; is effective only on an external reference variable.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1254C</th>
<th>__io operate on function declarator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__io&quot; cannot be specified to a function declarator. Assure that a valid type qualifier is specified. Continues the compilation neglecting the &quot;__io&quot;.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1255C</th>
<th>__io operate struct or union member</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__io&quot; is effective only on a variable. A part of members for a struct or union cannot be qualified by the &quot;__io&quot;. The &quot;__io&quot; is ignored.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1256C</th>
<th>__interrupt is specified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__interrupt&quot; is the extended specification.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1257C</th>
<th>__subinterrupt is specified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__subinterrupt&quot; is the extended specification.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1258C</th>
<th>__io is specified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__io&quot; is the extended specification.</td>
</tr>
</tbody>
</table>
APPENDIX

W1259C constant out of range of `register-constant'

[Explanation]
The value of an integral constant specified for the first argument of the "__regload" or the "__regstore" is larger than a number of registers.

W1260C declaration with obsolete modified typedef type

[Explanation]
A typedef declaration is made in an old style.

[Note]
This message is not used.

W1261C parameter type of function definition mismatch prototype: `%D'

[Explanation]
Though no parameter is declared in a function definition, a non-void parameter is declared in a prototype declaration of the function. Modify the parameter declaration.

W1262C function `%D' is defined here but it is builtin symbol

[Explanation]
A function which is registered as built-in function is defined. Assure that the function name is correct.

W1263C plain `char' type value used as subscript

[Explanation]
A variable of a simple "char" type is specified as a subscript of an array. Note that how the type is recognized depends on systems and options specified.

W1264C switch clause without `default'

[Explanation]
No default label is specified in a switch statement. Assure that case labels are correct.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1265C</td>
<td>pointer type required for argument: %D</td>
<td>An argument which specifies a format character string is not the pointer type. The argument is not checked.</td>
</tr>
<tr>
<td>W1266C</td>
<td>no struct members have name</td>
<td>None of members for a struct has a name. At least one must have a name.</td>
</tr>
<tr>
<td>W1267C</td>
<td>no union members have name</td>
<td>None of members for a union has a name. At least one must have a name.</td>
</tr>
<tr>
<td>W1268C</td>
<td>shift count %X is negative: %O</td>
<td>The integral constant specified as a shift value of a shift operation is negative. The shift operation is performed badly.</td>
</tr>
<tr>
<td>W1269C</td>
<td>shift count %X is too large for <code>%T</code>: %O</td>
<td>The integral constant specified as a shift value of a shift operation is larger than the size of the type for an operand.</td>
</tr>
<tr>
<td>W1270C</td>
<td>size of member <code>^%D</code> is zero</td>
<td>The size of a member must be positive.</td>
</tr>
<tr>
<td>W1271C</td>
<td>`%T' is used</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The &quot;int&quot;, &quot;signed int&quot; or &quot;unsigned int&quot; type is specified. The size of the int type differs for systems, then it is recommended to specify &quot;short&quot; or &quot;long&quot; simultaneously.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1272C</th>
<th><code>%T' is used for declaration </code>%D'</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td>The &quot;int&quot; or &quot;unsigned int&quot; type is specified. The size of the int type differs for systems, then it is recommended to specify &quot;short&quot; or &quot;long&quot; simultaneously.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1273C</th>
<th>comparison between pointer and constant: %O</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td>In a comparative operation, a pointer and an integral constant are compared. If an operand of the operation is the pointer, it needs to be matched with the other.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1274C</th>
<th>comparison between NULL and address of data object or function: %O</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td>A data object or an address of a function is compared with a null pointer constant. A data object or an address of a function cannot be a null, so the result of the comparison is determined at the compilation.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1275C</th>
<th>assignment from <code>%T' to </code>%T'</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td>An assignment operation to the &quot;int&quot; or &quot;unsigned int&quot; type is performed. The size of the int type differs for systems, then it is recommended to specify &quot;short&quot; or &quot;long&quot; simultaneously.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1276C</th>
<th>cast expression cannot be lvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td>According to the ANSI, a cast expression cannot be an lvalue (an expression as a destination of the assignment operation). Cast the right-hand side value to the type of the left-hand side value if possible.</td>
<td></td>
</tr>
</tbody>
</table>
W1277C  size of casting type is wider: type conversion from `%T' to `%T'

[Explanation]
The size of a destination type for a cast operation is larger than that of the type for an operand. The resulting value of the pointer may be wrong.

W1278C  size of casting type is narrower: type conversion from `%T' to `%T'

[Explanation]
The size of a destination type for a cast operation is smaller than that of the type for an operand. The resulting value of the pointer may be wrong.

W1279C  `long double' type specifier used for `%D'

[Explanation]
The long double type is specified.

[Note]
This message is not used.

W1280C  `long double' used: %O

[Explanation]
The long double type is specified.

[Note]
This message is not used.

W1281C  comparison between pointers: %O

[Explanation]
A comparison between pointers.

[Note]
This message is not used.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1282C</td>
<td>misalign declaration of <code>double'/'long double' struct/union member: </code>%D'</td>
<td>The boundary alignment of a member for the double or long double type is invalid.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1283C</td>
<td>layout of bit-field depends on application binary interface: `%D'</td>
<td>The layout of a bit-field depends on the application binary interface.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1284C</td>
<td>bit-field declaration without signed and unsigned: `%D'</td>
<td>A bit-field is declared without signed or unsigned specified.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1285C</td>
<td>bit-field declaration with enumerate type: `%D'</td>
<td>A bit-field of the enumeration type is declared.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1286C</td>
<td>useless type qualifier of <code>%T</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A type qualifier specified with the &quot;void&quot; type is meaningless. Ignore the type qualifier.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1287C</td>
<td>invalid void type parameter declaration: <code>%D</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A void type is invalid in a parameter declaration.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1288C</td>
<td>too large object <code>%D</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The size of the object is too large.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1289C</td>
<td>`...' in expression violates ANSI C specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The `...' can be specified only in a parameter list of a function declaration. Modify the expression.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1290C</td>
<td>too long multi-character character constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Too many characters are specified in a character constant to express by a word. Continues the compilation making the beginning part of a word size valid.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX

W1291C multi-character character constant specified

[Explanation]
More than one character constant is specified in a character constant. Continues the compilation making all of the specified characters constant valid.

W1292C directive line syntax error: newline expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system. An invalid character is found at the end of such a directive. The line feed character must be specified immediately after the directives. Continues the compilation neglecting the character.

W1293C #pragma: unknown `\%s`

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system. Though the "#pragma" is one of those directives, the indicated one is an unknown kind. Continues the compilation neglecting the directive.

W1294C #pragma int_to_unsigned: syntax error: identifier is expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system. Though the "#pragma int_to_unsigned" is one of those directives, an identifier is necessary for a parameter of the pragma. Continues the compilation neglecting the "#pragma" directive.

W1295C #pragma int_to_unsigned: type of identifier have to be function type returning unsigned int

[Explanation]
An identifier specified in the "#pragma int_to_unsigned" is not a function of the unsigned int type. The pragma is ineffective.
W1296C  #pragma optimized: optimization option is not specified

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.
Though the "#pragma optimized" is one of those directives, this pragma needs an optimization option. Continues the compilation neglecting the "#pragma" directive.

[Note]
This message is not used.

W1297C  #pragma optimized: on or off are expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.
Though the "#pragma optimized" is one of those directives, a parameter for this pragma must be an identifier. Continues the compilation neglecting the "#pragma" directive.

[Note]
This message is not used.

W1298C  #pragma: unknown switch

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.
Though the "#pragma optimized" is one of those directives, a parameter for this pragma must be one of `on', `off' or `default'. Continues the compilation neglecting the "#pragma" directive.

[Note]
This message is not used.
## APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1299C</td>
<td><code>#pragma: invalid switch specified</code></td>
</tr>
</tbody>
</table>

**[Explanation]**

Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.

Though the "`#pragma optimized`" is one of those directives, the specified parameter for the pragma is meaningless. Continues the compilation neglecting the "`#pragma`" directive.

**[Note]**

This message is not used.

| W1300C   | `#pragma: optimization specifier required`  |

**[Explanation]**

Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.

Though the "`#pragma global`, `#pragma statement`, `#pragma loop` and `#pragma procedure" are among those directives, these pragma need an optimization option. Continues the compilation neglecting the "`#pragma`" directive.

**[Note]**

This message is not used.

| W1301C   | `#pragma ident: syntax error: string constant is expected` |

**[Explanation]**

Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.

Though the "`#pragma ident" is one of those directives, this pragma needs a string literal for a parameter. Continues the compilation neglecting the "`#pragma`" directive.

| W1302C   | `#line: syntax error: digit required`       |

**[Explanation]**

Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.

Though the "`#line" is one of those directives, this directive needs a number for a line number. Continues the compilation neglecting the "`#line`" directive.
Appendix E  ERROR MESSAGE

W1303C  #ident: syntax error: string constant expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some
directives are processed by the compiler owing to the structure of the system.
Though the "#ident" is one of those directives, this directive needs a string literal. Continues
the compilation neglecting the "#ident" directive.

W1304C  #pragma weak: syntax error: `=' is expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some
directives are processed by the compiler owing to the structure of the system.
Though the "#pragma weak" is one of those directives, identifiers that are specified in this
pragma must be connected with `='. Continues the compilation neglecting the "#pragma"
directive.

W1305C  #pragma weak: syntax error: identifier is expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some
directives are processed by the compiler owing to the structure of the system.
Though the "#pragma weak" is one of those directives, this pragma needs identifiers to be
specified. Continues the compilation neglecting the "#pragma" directive.

W1306C  #pragma echo: syntax error: string constant is expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some
directives are processed by the compiler owing to the structure of the system.
Though the "#pragma echo" is one of those directives, this pragma needs a string literal for
a parameter. Continues the compilation neglecting the "#pragma" directive.

W1307C  #assert directive used

[Explanation]
The "#assert" directive is specified.

[Note]
This message is not used.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1308C</td>
<td>#error directive used</td>
<td>The &quot;#error&quot; directive is specified.</td>
</tr>
<tr>
<td>W1309C</td>
<td>#pragma locale: syntax error: string constant is expected</td>
<td>Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system. Though the &quot;#pragma locale&quot; is one of those directives, this pragma needs a string literal for a parameter. Continues the compilation neglecting the &quot;#pragma&quot; directive.</td>
</tr>
<tr>
<td>W1310C</td>
<td>#sccs directive used</td>
<td>The &quot;#sccs&quot; directive is specified.</td>
</tr>
<tr>
<td>W1311C</td>
<td>#ident directive used</td>
<td>Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system. Though the &quot;#ident&quot; is one of those directives, this directive is an old style. It is recommended to replace it by &quot;#pragma ident&quot;. Continues the compilation leaving it as it is.</td>
</tr>
<tr>
<td>W1312C</td>
<td>#pragma: unknown pragma</td>
<td>Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system. Though the &quot;#pragma&quot; is one of those directives, no pragma name is specified. Continues the compilation neglecting the &quot;#pragma&quot; directive.</td>
</tr>
</tbody>
</table>
W1313C  #pragma: syntax error: `(` expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.
Though the "#pragma unknown_control_flow" is one of those directives, the `(` must follow it if another token follows. Continues the compilation neglecting the "#pragma" directive.

W1314C  #pragma: syntax error: `)` or `;` expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.
Though the "#pragma unknown_control_flow" is one of those directives, the `(` or `)` is specified after an identifier. Continues the compilation neglecting the "#pragma" directive.

W1315C  #pragma: identifier expected

[Explanation]
Though a preprocessor directive is generally processed by the preprocessor, some directives are processed by the compiler owing to the structure of the system.
Though the "#pragma" directive is one of those directives, an identifier is needed to be specified just after the "pragma" directive. Continues the compilation neglecting the "#pragma" directive.

W1316C  #pragma: string constant expected

[Explanation]
A string literal is needed for a parameter of the "#pragma" directive.

[Note]
This message is not used.

W1317C  #pragma: unknown specifier

[Explanation]
An invalid parameter is specified in the "#pragma" directive.

[Note]
This message is not used.
## APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1318C</td>
<td>#pragma %s: unknown optimization specifier</td>
<td>An invalid parameter is specified in the &quot;#pragma&quot; directive.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1319C</td>
<td>#pragma %s: invalid argument</td>
<td>An invalid parameter is specified in the &quot;#pragma&quot; directive.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1320C</td>
<td>#pragma %s: identifier expected</td>
<td>An identifier is needed for a parameter of the &quot;#pragma&quot; directive.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1321C</td>
<td>#pragma %s: invalid token: identifier expected</td>
<td>An identifier is needed for a parameter of the &quot;#pragma&quot; directive.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>W1322C</td>
<td>#pragma %s: invalid token: too many parameters</td>
<td>Too many parameters are specified in the &quot;#pragma&quot; directive.</td>
<td></td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1323C</td>
<td><code>#pragma %s: integer constant expected</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] An integral constant is needed for a parameter of</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the &quot;#pragma&quot; directive.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1324C</td>
<td><code>#pragma %s: invalid token: integer constant expected</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] An integral constant is needed for a parameter of</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the &quot;#pragma&quot; directive.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1325C</td>
<td><code>#pragma %s: invalid optimization specifier</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] An invalid optimization specifier is specified in</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the &quot;#pragma&quot; directive.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1326C</td>
<td><code>#pragma section: syntax error: </code>=<code> is expected</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The <code>=</code> is needed just after the &quot;attr&quot; parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>or the &quot;locate&quot; parameter of the &quot;#pragma section&quot; directive.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Continues the compilation neglecting the &quot;#pragma section&quot;.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W1327C</td>
<td><code>#pragma section: invalid hexadecimal constant</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] An invalid value is set for the locate parameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>of the &quot;#pragma section&quot;.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note] This message is not used.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Explanation</td>
<td>Note</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>W1328C</td>
<td>#pragma section: invalid sectionname specified</td>
<td>An unknown identifier is specified for a section name of the &quot;#pragma section&quot;. Continues the compilation neglecting the &quot;#pragma section&quot;.</td>
<td></td>
</tr>
<tr>
<td>W1329C</td>
<td>#pragma section: invalid sectionattr specified</td>
<td>An invalid identifier is specified for the attr parameter of the &quot;#pragma section&quot;. Continues the compilation neglecting the &quot;#pragma section&quot;.</td>
<td></td>
</tr>
<tr>
<td>W1330C</td>
<td>#pragma asm: syntax error: unknown specifier</td>
<td>An excess character is found after the &quot;#pragma asm&quot;.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is not used.</td>
<td></td>
</tr>
<tr>
<td>W1331C</td>
<td>#pragma section: too long identifier is specified</td>
<td>Too long identifier is specified for a section name of the &quot;#pragma section&quot; . Continues the compilation making the first 255 bytes valid.</td>
<td></td>
</tr>
<tr>
<td>W1332C</td>
<td>#pragma asm: syntax error: <code>#pragma endasm</code> is not specified</td>
<td>The &quot;#pragma endasm&quot; is missing.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is not used.</td>
<td></td>
</tr>
</tbody>
</table>
W1333C #pragma section: syntax error: identifier is expected

[Explanation]
No identifier is specified for a section name of the "#pragma section". Continues the compilation neglecting the "#pragma section".

W1334C #pragma inline: syntax error: identifier is expected

[Explanation]
No identifier is specified for a function name of the "#pragma inline". Continues the compilation neglecting the "#pragma inline".

W1335C #pragma section: syntax error: unknown specifier

[Explanation]
An unknown parameter is specified other than a section name, an attribute or a location address in the "#pragma section". Continues the compilation neglecting the "#pragma section".

W1336C #pragma %s: syntax error: identifier is expected

[Explanation]
No interrupt function name is specified in the "#pragma intvect" or "#pragma defvect". Continues the compilation neglecting the "#pragma intvect" or "#pragma defvect".

W1337C #pragma intvect: invalid decimal constant

[Explanation]
An invalid interrupt number is specified in the "#pragma intvect".

[Note]
This message is not used.
### W1338C
#pragma %s: invalid type of interrupt function `%s`

**[Explanation]**
The type of the function specified in the "#pragma intvect" or "#pragma defvect" is not as __interrupt (*)(void). The "#pragma intvect" or "#pragma defvect" is ineffective.

### W1339C
#pragma intvect: vectornumber is not integral constant expression

**[Explanation]**
An invalid interrupt number is specified in the "#pragma intvect".

**[Note]**
This message is not used.

### W1340C
#pragma intvect: syntax error: vectornumber is expected

**[Explanation]**
No vector number is specified in the "#pragma intvect". Continue the compilation neglecting the "#pragma intvect".

### W1341C
#pragma %s: interrupt function `%s' is not found

**[Explanation]**
Function specified by "#pragma intvect" or "#pragma defvect" is not found. Continue the compilation regarding that the "#pragma intvect" or "#pragma defvect" is ineffective.

### W1342C
newline is expected

**[Explanation]**
Continue the compilation neglecting the character.

### W1343C
cannot open compiler message file "%s"

**[Explanation]**
A message built in the compiler is used in place of a message file.
Appendix E  ERROR MESSAGE

W1344C  cannot open compiler message file

[Explanation]
A message built in the compiler is used in place of a message file.

W1345C  %Z: invalid lvalue

[Explanation]
The increment or decrement operator is applied to a result of a cast operation. The result of
the cast operation is not an lvalue and cannot be an operand of those operators. Continue
the compilation neglecting the cast operation.

W1346C  #pragma defvect: duplicate defvect function

[Explanation]
A function specified in the "#pragma intvect" is also specified in the "#pragma defvect". Continue
the compilation neglecting the latter directive.

W1347C  type `%T' defined as function return type: function `%D'

[Explanation]
A struct or union type is specified as a return value of a function with a tag declaration. Continue
the compilation leaving it as it is.

W1349C  %Z incompatible pointer types: expected `%T' actual `%T' in function `%D'

[Explanation]
The type of the return value for a function and that of an expression in a return statement
are incompatible. Modify the expression in the return statement to match with the type of
the return value.

W1350C  %Z incompatible pointer types: expected `%T' actual `%T': argument %d of `%D'

[Explanation]
The type of an argument in a function call is incompatible with that of a parameter declared. Modify the argument to match with the parameter type.
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1351C</td>
<td><code>%Z incompatible pointer types: expected </code>%T' actual `%T': argument %d</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The type of an argument in a function call is incompatible with that of</td>
</tr>
<tr>
<td></td>
<td>a parameter declared. Modify the argument to match with the parameter</td>
</tr>
<tr>
<td></td>
<td>type.</td>
</tr>
<tr>
<td>W1352C</td>
<td><code>%Z incompatible pointer types from </code>%T' to <code>%T': </code>%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Types of an initializer and of a target are incompatible. Modify the</td>
</tr>
<tr>
<td></td>
<td>initializer to match with the target type.</td>
</tr>
<tr>
<td>W1353C</td>
<td><code>%Z incompatible pointer types from </code>%T' to `%T': %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Types of both sides of the assignment operation are incompatible.</td>
</tr>
<tr>
<td></td>
<td>Modify the right-hand side value to match with the type of the left-</td>
</tr>
<tr>
<td></td>
<td>hand side value.</td>
</tr>
<tr>
<td>W1354C</td>
<td><code>%Z: constant out of range for </code>%T' bit-field: its width %d</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>A constant specified as an initializer exceeds the range of a bit-field.</td>
</tr>
<tr>
<td></td>
<td>Modify the constant value.</td>
</tr>
<tr>
<td>W1355C</td>
<td>multiple `__far' specified</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Several &quot;__far&quot;, which is effective on a symbol or a pointer, are</td>
</tr>
<tr>
<td></td>
<td>specified. Only one is effective.</td>
</tr>
<tr>
<td>W1356C</td>
<td>multiple `__near' specified</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Several &quot;__near&quot;, which is effective on a symbol or a pointer, are</td>
</tr>
<tr>
<td></td>
<td>specified. Only one is effective.</td>
</tr>
</tbody>
</table>
appendix e error message

w1357c  multiple `__direct' specified

[explanation]
several "__direct", which is effective on a symbol or a pointer, are specified. only one is effective.

w1358c  typedefed type already qualified with `__far'

[explanation]
though "__far" is specified in a typedef declaration, it is specified together with the typedef name. the effect of the "__far" is the same as one "__far" is specified.

w1359c  typedefed type already qualified with `__near'

[explanation]
though "__near" is specified in a typedef declaration, it is specified together with the typedef name. the effect of the "__near" is the same as one "__near" is specified.

w1360c  typedefed type already qualified with `__direct'

[explanation]
though "__direct" is specified in a typedef declaration, it is specified together with the typedef name. the effect of the "__direct" is the same as one "__direct" is specified.

w1361c  __near and __far can not be specified at the same time

[explanation]
The "__far" and "__near" that are effective on a symbol or a pointer cannot be specified simultaneously. one specified afterward is ignored.

w1362c  __far operate struct or union member

[explanation]
The "__far" is specified to characterize a variable or a pointer. it is not applicable to a member of a struct or union. continues the compilation neglecting the "__far".
### APPENDIX

<table>
<thead>
<tr>
<th>W1363C</th>
<th>__near operate struct or union member</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__near&quot; is specified to characterize a variable or a pointer. It is not applicable to a member of a struct or union. Continues the compilation neglecting the &quot;__near&quot;.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1364C</th>
<th>__far is specified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__far&quot; is the extended specification.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1365C</th>
<th>__near is specified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__near&quot; is the extended specification.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1366C</th>
<th>__direct is specified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__direct&quot; is the extended specification.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1367C</th>
<th>__direct operate on function declarator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__direct&quot; is not applicable to a function declarator. Check the specification of the type qualifier. Continues the compilation neglecting the &quot;__direct&quot;.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W1368C</th>
<th>__direct operate struct or union member</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;__direct&quot; is effective only on a variable. It is not applicable to a member of a struct or union. Continues the compilation neglecting the &quot;__direct&quot;.</td>
</tr>
</tbody>
</table>
W1369C  __io and __direct can not be specified at the same time

[Explanation]
The "__io" and "__direct" are type qualifiers which specify a region where a variable is allocated. Those cannot be specified simultaneously. One specified afterward is ignored.

W1370C  %Z: type conversion from `%T` to `%T': `%D'

[Explanation]
No correct value can be expected because an information of a segment for a pointer cannot be attached. Do not cast from a __near pointer to a __far pointer in a constant expression of an initializer.

W1371C  __far or __near cannot specify to local variable

[Explanation]
Local variables are allocated in the same stack area, so the "__far" and "__near" type qualifiers are meaningless. Continues the compilation neglecting the "__far" or "__near".

W1372C  first argument is not integral constant expression

[Explanation]
The first argument for the "__regload" or "__regstore" is not an integral constant.

[Note]
This message is not used.

W1373C  indirect reference function call specified

[Explanation]
A function is called through a function pointer.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1374C</td>
<td><code>#pragma %s: already exist</code></td>
</tr>
<tr>
<td></td>
<td><strong>Explanation</strong></td>
</tr>
<tr>
<td></td>
<td>The indicated “#pragma” is already specified. It cannot be nested. Continues the compilation neglecting the indicated pragma.</td>
</tr>
<tr>
<td>W1375C</td>
<td><code>#pragma %s' expected</code></td>
</tr>
<tr>
<td></td>
<td><strong>Explanation</strong></td>
</tr>
<tr>
<td></td>
<td>The indicated “#pragma” is not specified. Continues the compilation assuming that the pragma is specified at the end of the compile unit.</td>
</tr>
<tr>
<td>W1376C</td>
<td><code>#pragma intvect: invalid constant for vector number</code></td>
</tr>
<tr>
<td></td>
<td><strong>Explanation</strong></td>
</tr>
<tr>
<td></td>
<td>An invalid value is specified for a vector number of the &quot;#pragma intvect&quot;. Continues the compilation neglecting the &quot;#pragma intvect&quot;.</td>
</tr>
<tr>
<td>W1377C</td>
<td><code>#pragma intvect: invalid constant for mode number</code></td>
</tr>
<tr>
<td></td>
<td><strong>Explanation</strong></td>
</tr>
<tr>
<td></td>
<td>An invalid value is specified for a mode of the &quot;#pragma intvect&quot;. Continues the compilation neglecting the mode specification.</td>
</tr>
<tr>
<td>W1378C</td>
<td><code>#pragma intvect: mode number is out of range</code></td>
</tr>
<tr>
<td></td>
<td><strong>Explanation</strong></td>
</tr>
<tr>
<td></td>
<td>The value of a mode specified in the &quot;#pragma intvect&quot; is larger than 255. Continues the compilation neglecting the mode specification.</td>
</tr>
<tr>
<td>W1379C</td>
<td><code>#pragma intvect: difference mode number for existed same vector number</code></td>
</tr>
<tr>
<td></td>
<td><strong>Explanation</strong></td>
</tr>
<tr>
<td></td>
<td>Though the &quot;#pragma intvect&quot; is already specified for the same function name with the same vector number, a mode is specified differently. Continues the compilation neglecting the &quot;#pragma intvect&quot;.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>W1380C</td>
<td><code>#pragma section: invalid constant for allocated address</code></td>
</tr>
</tbody>
</table>
|            | [Explanation]  
|            | An invalid value is specified for an allocation address of a locate parameter in the "#pragma section".  
|            | Continues the compilation neglecting the "#pragma section". |
| W1381C     | `__nosavereg is specified` |
|            | [Explanation]  
|            | The "__nosavereg" is the extended specification. |
| W1382C     | `multiple `__nosavereg' specified` |
|            | [Explanation]  
|            | Several "__nosavereg", which is effective on a symbol or a pointer, are specified. Only one is effective. |
| W1383C     | `typedefed type already qualified with `__nosavereg'` |
|            | [Explanation]  
|            | Though the "__nosavereg" is specified in a typedef declaration, it is specified together with the typedef name. The effect of the "__nosavereg" is the same as one "__nosavereg" is specified. |
| W1384C     | `__nosavereg not operate on function declarator` |
|            | [Explanation]  
|            | The "__nosavereg" is a type qualifier effective only on a function declarator. Modify the type specifier. Continues the compilation neglecting the "__nosavereg". |
| W1385C     | `cannot initialize variable with `__io'` |
|            | [Explanation]  
<p>|            | An initial value cannot be set in the area where a variable which is qualified by the &quot;__io&quot; is allocated. Continues the compilation making the initialization invalid. |</p>
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1386C</td>
<td>different size: argument type <code>%T' with prototype </code>%T': argument %d</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The widths of types of a parameter specified in a prototype declaration and of an argument do not correspond. The type specified in the prototype declaration is adopted.</td>
</tr>
<tr>
<td>W1387C</td>
<td>different unsignedness: argument type <code>%T' with prototype </code>%T': argument %d</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The signs for types of a parameter specified in a prototype declaration and of an argument do not correspond. The type specified in the prototype declaration is adopted.</td>
</tr>
<tr>
<td>W1388C</td>
<td>Ignore since the 255th character of symbol `%s'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The external symbol name is too long. The first 254 characters are valid.</td>
</tr>
<tr>
<td>W1389C</td>
<td>`#pragma intvect': vector number is out of range</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The vector number specified in the &quot;#pragma intvect&quot; exceeds the maximum value in this system. Continues the compilation assuming that no value is specified.</td>
</tr>
<tr>
<td>W1390C</td>
<td>restrict is specified</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The &quot;restrict&quot; is the extended specification.</td>
</tr>
<tr>
<td>W1391C</td>
<td>multiple `restrict' specified</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>&quot;restrict&quot; specified by the declaration of the typedef name is used together with the typedef name. The effect of all of them is equal to that of one &quot;restrict&quot;.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>W1392C</td>
<td>typedefed type already qualified with <code>restrict</code></td>
</tr>
</tbody>
</table>

[Explanation]
Though the "restrict" is specified in a typedef declaration, it is specified together with the typedef name. The effect of the "restrict" is the same as one "restrict" is specified.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1393C</td>
<td>%Z: exist data without initial value: <code>%D</code></td>
</tr>
</tbody>
</table>

[Explanation]
A data which is a member of a struct or an array is not initialized. Continues the compilation assuming that 0 is set for the initial value.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4001C</td>
<td>type of function designator must be function type</td>
</tr>
</tbody>
</table>

[Explanation]
A type other than the function type nor the function pointer type is specified in the function call.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4002C</td>
<td>pointer or array type are expected</td>
</tr>
</tbody>
</table>

[Explanation]
A type other than the pointer type nor the array type is specified as pointer reference (`*` operator).

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4003C</td>
<td>pointer to struct/union type required for left operand</td>
</tr>
</tbody>
</table>

[Explanation]
Specify a pointer to the struct or union.

[Note]
This message is not used.
APPENDIX

E4004C  struct/union type required for left operand

[Explanation]
Specify struct or union type.

[Note]
This message is not used.

E4005C  scalar type required

[Explanation]
A type other than the scalar type (arithmetic or pointer type) is specified as an operand of the increment, decrement, or "!" operator.

E4006C  integral type required

[Explanation]
A type other than the integral type is specified as an operand of the operator.

E4007C  arithmetic type required

[Explanation]
A type other than the arithmetic type (integral or floating point type) is specified as an operand of the unary "+" or "-" operator.

E4008C  operation and assignment from pointer type to integral type

[Explanation]
The integral type is specified on the left-hand side of the "+=" or "-=" operator and the pointer type on the right-hand side.
Though the operator "+=" adds the right-hand side value to the left-hand side value and substitutes the result to the left-hand side, an addition of the integral type and the pointer type results in the pointer type,
then it does not match with the left-hand side.
Meanwhile the subtraction of the pointer type from the integral type is invalid.
### Appendix E ERROR MESSAGE

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4009C</td>
<td>unknown size of incomplete type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The struct, union or enumeration tag has no member, and no size information is available.</td>
</tr>
<tr>
<td>E4010C</td>
<td>unknown size of array type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>A type with an unknown size is specified as an element of an array. Cannot determine the total size of the array.</td>
</tr>
<tr>
<td>E4011C</td>
<td>unknown size of function type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Attempted to get the size of the function type. It's impossible, so 1 is assumed.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4012C</td>
<td>unknown size</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Attempted to get the size of the void type. It's impossible, so 1 is assumed.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4013C</td>
<td>invalid null subscript of array type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Attempted to get the size of the array type with unknown size. 0 is assumed.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>E4014C</td>
<td>invalid variable subscript of array</td>
</tr>
<tr>
<td>E4015C</td>
<td>invalid label</td>
</tr>
<tr>
<td>E4016C</td>
<td>undefined label <code>%D</code> is used as destination</td>
</tr>
<tr>
<td>E4017C</td>
<td>label <code>%D</code> redefined</td>
</tr>
<tr>
<td>E4018C</td>
<td><code>%Z: too many initializers for </code>%T': `%D'</td>
</tr>
<tr>
<td>E4019C</td>
<td><code>%Z: incomplete type </code>%T' specified: `%D'</td>
</tr>
</tbody>
</table>
Appendix E  ERROR MESSAGE

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4020C</td>
<td>%Z: invalid array null dimension: <code>%D</code></td>
<td>Attempted to initialize an array type which has as its element an array type with no subscript specified. For initialization, the subscript only of the left end element (next to the symbol) can be omitted.</td>
</tr>
<tr>
<td>E4021C</td>
<td>%Z: invalid array variable dimension: <code>%D</code></td>
<td>Attempted to initialize an array type which has as its element an array type with non-integral-constant subscript. The subscript of an array must be an integral constant.</td>
</tr>
<tr>
<td>E4022C</td>
<td>%Z: invalid initializer: <code>%D</code></td>
<td>An object of the function type cannot be specified as an initializer.</td>
</tr>
<tr>
<td>E4023C</td>
<td>%Z: invalid initializer for array: <code>%D</code></td>
<td>_INITIALIZER is badly specified for an array.</td>
</tr>
<tr>
<td>E4024C</td>
<td>%Z: invalid initializer for function: <code>%D</code></td>
<td>Initializer is specified for the function declaration. A function cannot be initialized.</td>
</tr>
<tr>
<td>E4025C</td>
<td>%Z: invalid initializer for struct/union: <code>%D</code></td>
<td>The initializer for the struct or union is specified in a bad manner.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>E4026C</td>
<td><code>%Z: invalid initializer for </code>%T': <code>%D'</code></td>
<td></td>
</tr>
</tbody>
</table>

[Explanation]
Only a symbol must be specified to initialize a struct or union with another struct or union of a compatible type.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4027C</td>
<td><code>%Z: invalid initializer for struct/union</code></td>
</tr>
</tbody>
</table>

[Explanation]
The type of the struct or union specified as an initializer is incompatible with that of the target struct or union.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4028C</td>
<td><code>%Z: too many initializers for array: </code>%D'`</td>
</tr>
</tbody>
</table>

[Explanation]
More initializers are specified than the subscript value of array.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4029C</td>
<td><code>%Z: too deeply nested brace: </code>%D'`</td>
</tr>
</tbody>
</table>

[Explanation]
Too many braces are specified to enclose the initializer. Modify to fit for the type of the target.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4030C</td>
<td><code>%Z: integral constant expression required for bit-field </code>%D': <code>%D'</code></td>
</tr>
</tbody>
</table>

[Explanation]
Only an integral constant can be specified as an initializer of a bit-field.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4031C</td>
<td><code>%Z: too many initializers for an element: </code>%D'`</td>
</tr>
</tbody>
</table>

[Explanation]
Too many initializers are specified. Only one initializer is needed for one variable or member.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4032C</td>
<td><code>%Z: constant expression is expected for %s: </code>%D'`</td>
<td>Only a constant expression can be specified as an initializer.</td>
</tr>
<tr>
<td>E4033C</td>
<td><code>=</code> expected before initializer</td>
<td>The compound assignment operator is specified prior to assigning an initial value. Only a simple assignment operator can be used in the initialization.</td>
</tr>
<tr>
<td>E4034C</td>
<td><code>=</code> expected before enumerator initializer</td>
<td>The compound assignment operator is specified prior to initializing an enumerator. Only a simple assignment operator can be used in an initialization.</td>
</tr>
<tr>
<td>E4035C</td>
<td>`continue' found outside loops</td>
<td>A continue statement is outside the loop. The statement means to jump to the end of the innermost loop which encloses the statement, so it must be inside a loop.</td>
</tr>
<tr>
<td>E4036C</td>
<td>`break' found outside switchs and loops</td>
<td>A break statement is outside the loop or the switch statement. The statement means to jump to the end of the innermost loop or switch statement which encloses the statement, so it must be inside a loop or switch statement.</td>
</tr>
<tr>
<td>E4037C</td>
<td>unterminated string literal: `&quot;' expected</td>
<td>The string literal does not end.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td>This message is not used.</td>
</tr>
</tbody>
</table>
APPENDIX

E4038C  EOF in comment

[Explanation]
The EOF is found unless the comment block ends. The "*/" is missing to terminate the
comment block.

E4039C  EOF in string literal

[Explanation]
The EOF is found unless the string literal ends. The double quotation is missing to
terminate the string literal.

E4040C  sorry, internal limitation: quoted character too long

[Explanation]
The length of the string constant or the hexadecimal beginning with "\x" in the string literal
exceeds the limit of this system, 4028.

E4041C  character constant has no character expression

[Explanation]
No character is found in the character constant.

E4042C  unterminated character constant: newline in character constant

[Explanation]
The newline character is found in the character constant. The single quotation is missing to
terminate the character constant.

E4043C  unterminated character constant: EOF in character constant

[Explanation]
The EOF is found in the character constant. The single quotation is missing to terminate
the character constant.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4044C</td>
<td>too many postfix characters <code>%c</code> for constant</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The postfix of the numerical constant is not specified correctly.</td>
</tr>
<tr>
<td>E4045C</td>
<td>numeric octal constant contains invalid character</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>A character which is not an octal digit is used in the octal constant.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4046C</td>
<td>binary constant cannot be floating point constant</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Binary constant beginning with <code>0b</code> or <code>0B</code> cannot be floating point constant.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4047C</td>
<td>invalid postfix character <code>%c</code> after integer constant</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>An alphabet which cannot be recognized as a postfix is specified at the end of the integral constant.</td>
</tr>
<tr>
<td>E4048C</td>
<td>invalid postfix character <code>%c</code> for radix %d</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>A character which is not an alphabet and cannot be recognized as a postfix is specified at the end of the integral constant.</td>
</tr>
</tbody>
</table>
**APPENDIX**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Explanation</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4049C</td>
<td>no digits of floating exponent part</td>
<td>No digit is specified for the exponent part of the floating point number.</td>
<td></td>
</tr>
<tr>
<td>E4050C</td>
<td>hexadecimal constant cannot be floating point constant</td>
<td>The `.' is found in the hexadecimal constant.</td>
<td></td>
</tr>
<tr>
<td>E4051C</td>
<td>invalid postfix character <code>%c</code> after floating point constant</td>
<td>An invalid character for postfix of floating point number is specified at the end of the floating point constant.</td>
<td></td>
</tr>
<tr>
<td>E4052C</td>
<td>invalid token: <code>..</code></td>
<td>An invalid token <code>..</code> is found. The <code>.' is an operator, the </code>...' an ellipsis.</td>
<td></td>
</tr>
<tr>
<td>E4053C</td>
<td>integer constant out of range</td>
<td>The integral constant cannot be expressed internally.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4054C</td>
<td>invalid character <code>%c</code></td>
<td>An invalid character is found. It is not included in the source character set.</td>
<td></td>
</tr>
</tbody>
</table>
Appendix E  ERROR MESSAGE

E4055C  invalid binary constant

[Explanation]
The first digit of the binary constant beginning with `0b' or `0B' is not 0 nor 1.

[Note]
This message is not used.

E4056C  invalid hexadecimal constant

[Explanation]
The first digit of the hexadecimal constant beginning with `0x' or `0X' is not a hexadecimal digit.

E4057C  invalid multibyte character constant

[Explanation]
There is an invalid multi-byte character which cannot be recognized as a character code written in the wide-character constant.

E4058C  invalid multibyte string literal

[Explanation]
There is an invalid multi-byte character which cannot be recognized as a character code written in the wide string literal.

E4059C  invalid character `\%o' (octal)

[Explanation]
An invalid character is found. It is not included in the source character set.

E4060C  %s near wide character string constant

[Explanation]
This is a supplemental message for syntax error: A syntax error occurred near the wide-character string literal.
[Explanation]
This is a supplemental message for syntax error: A syntax error occurred near the wide-character constant.

E4062C  %s near `%s'

[Explanation]
This is a supplemental message for syntax error: A syntax error occurred near the indicated character.

E4063C  %s detected

[Explanation]
This is a supplemental message for syntax error: A syntax error occurred.

E4064C  %s near character constant

[Explanation]
This is a supplemental message for syntax error: A syntax error occurred near the character constant.

E4065C  %s near string constant

[Explanation]
This is a supplemental message for syntax error: A syntax error occurred near the string literal.

E4066C  division by 0

[Explanation]
The constant operation contains the division by 0.
Appendix E ERROR MESSAGE

**E4067C** invalid use of void type expression

[Explanation]
An expression of the void type is specified. No operation can be done on the void type.

**E4068C** incompatible subscript between variable and constant

[Explanation]
Subscripts of array are declared differently. Regard that types are incompatible.

[Note]
This message is not used.

**E4069C** incompatible subscript between variables

[Explanation]
Subscripts of array are declared differently. Regard that types are incompatible.

[Note]
This message is not used.

**E4070C** cannot concatenate character string literal and wide character string literal

[Explanation]
Cannot concatenate a character string literal and a wide character string literal. Types must correspond to concatenate.

**E4071C** undefined field `%I`

[Explanation]
A symbol referred as a member is not declared as a member.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4072C</td>
<td>different data types specified for declaration of <code>%s</code></td>
<td>Different data type specifiers are specified at once. Only one can be specified.</td>
</tr>
<tr>
<td>E4073C</td>
<td>invalid short or long specified with char for declaration of <code>%s</code></td>
<td>The type &quot;char&quot; cannot be specified with &quot;short&quot; nor &quot;long&quot;.</td>
</tr>
<tr>
<td>E4074C</td>
<td>invalid short or long specified for declaration of <code>%s</code></td>
<td>The type specifier cannot be specified with &quot;short&quot; nor &quot;long&quot;.</td>
</tr>
<tr>
<td>E4075C</td>
<td>invalid signed or unsigned specified for declaration of <code>%s</code></td>
<td>The type specifier cannot be specified with &quot;signed&quot; nor &quot;unsigned&quot;.</td>
</tr>
<tr>
<td>E4076C</td>
<td>multiple storage class specifier specified for declaration of <code>%s</code></td>
<td>Several storage class specifiers are written. Only one can be specified.</td>
</tr>
<tr>
<td>E4077C</td>
<td>invalid long specified for declaration of <code>%s</code></td>
<td>The &quot;long&quot; is coupled with an invalid type specifier.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>E4078C</td>
<td>invalid declaration specifier of both fundamental type and elaborate type for declaration of <code>%s</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type specifiers of the arithmetic type and of the elaborate type (struct, union or enumeration) are specified simultaneously.</td>
<td></td>
</tr>
<tr>
<td>E4079C</td>
<td>invalid declaration specifier of elaborate types for declaration of <code>%s</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Several type specifiers of the elaborate type (struct, union or enumeration) are specified.</td>
<td></td>
</tr>
<tr>
<td>E4080C</td>
<td>invalid struct/union definition: zero size structure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No valid member is declared in a tag of the struct or union. At least one member must be declared.</td>
<td></td>
</tr>
<tr>
<td>E4081C</td>
<td>invalid declaration of <code>%D</code> as parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The declaration of the parameter is invalid.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
<td></td>
</tr>
<tr>
<td>E4082C</td>
<td>redeclaration of <code>%D</code> in parameter list</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A symbol is redefined in the parameter list.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX

E4083C `%E %I' was declared as `%T' previously

[Explanation]
The tag name of the struct, union or enumeration is already declared as a different kind tag.
This is a supplemental message for E4179C.

E4084C function `%D' cannot be struct/union member

[Explanation]
A symbol of the function type cannot be declared as a member of the struct or union.

E4085C redeclaration of `%D': different object types

[Explanation]
A symbol is redeclared with several different object types (a feature of a symbol) in the same name space.

E4086C redeclaration of `%D'

[Explanation]
A symbol is redefined. Inspect the type, the linkage and the position declared.

E4087C linkage conflict declaration of `%D'

[Explanation]
The linkage of the symbol does not match with that defined previously.

E4088C invalid storage class specifier `%A' at %S specified for function `%D'

[Explanation]
The indicated storage class specifier cannot be used in a function declaration.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4089C</td>
<td>invalid storage class specifier <code>%A' at %S specified for </code>%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The indicated storage class specifier cannot be used in a declaration. Inspect the relation</td>
</tr>
<tr>
<td></td>
<td>between the specifier and the position it is declared.</td>
</tr>
<tr>
<td>E4090C</td>
<td>invalid initializer for declaration of `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>An initializer is specified in the declaration where an initialization is invalid.</td>
</tr>
<tr>
<td>E4091C</td>
<td>invalid type of array subscript: integral type expected</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Only the integral type is allowed for a subscript of the array.</td>
</tr>
<tr>
<td>E4092C</td>
<td>both separate parameter declaration and parameter list declaration are used</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Though a function is defined with a prototype declaration, non-prototype parameters are also declared. The separate parameter declaration is not needed.</td>
</tr>
<tr>
<td>E4093C</td>
<td>invalid constant expression in array subscript</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The expression specified in a subscript of an array is invalid. Specify a valid integral constant expression.</td>
</tr>
<tr>
<td>E4094C</td>
<td>invalid array subscript: integral constant expression is expected</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Only an integral constant expression can be specified as a subscript of the array. Specify a valid constant expression.</td>
</tr>
</tbody>
</table>
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4095C</td>
<td>subscript of array is negative</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A negative value is specified as a subscript of the array. Cannot determine the size of the array.</td>
</tr>
<tr>
<td>E4096C</td>
<td>array type of functions</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The element of an array is the function type, not a proper type for an element.</td>
</tr>
<tr>
<td>E4097C</td>
<td>array type of voids</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The element of an array is the void type, not a proper type for an element.</td>
</tr>
<tr>
<td>E4098C</td>
<td>function cannot return array</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The return value of the function is the array type. A function cannot return an array.</td>
</tr>
<tr>
<td>E4099C</td>
<td>function cannot return function</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The return value of the function is the function type. A function cannot return a function.</td>
</tr>
<tr>
<td>E4100C</td>
<td>parameter name is not specified</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A parameter name is needed in the function definition of the prototyped function.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>E4101C</td>
<td>invalid storage class specifier <code>%A' specified for function definition of </code>%D'</td>
</tr>
<tr>
<td>Explanation</td>
<td>The storage class specifier indicated cannot be used in function definition.</td>
</tr>
<tr>
<td>E4102C</td>
<td>function type is required for `%D'</td>
</tr>
<tr>
<td>Explanation</td>
<td>In the form of the function declaration where a function body follows a declarator, the symbol must be the function type.</td>
</tr>
<tr>
<td>E4103C</td>
<td>different data type declaration of `%D'</td>
</tr>
<tr>
<td>Explanation</td>
<td>The symbol defined as a function is already declared not as a function.</td>
</tr>
<tr>
<td>E4104C</td>
<td>redeclaration of parameter `%D'</td>
</tr>
<tr>
<td>Explanation</td>
<td>The symbol indicated is already declared as a parameter.</td>
</tr>
<tr>
<td>E4105C</td>
<td>parameter `%D' is missing in parameter list</td>
</tr>
<tr>
<td>Explanation</td>
<td>The parameter declared in the function definition without prototype declaration is not found in the list of identifiers in the function declarator.</td>
</tr>
<tr>
<td>E4106C</td>
<td>invalid void type parameter declaration in parameter list</td>
</tr>
<tr>
<td>Explanation</td>
<td>The void type is specified at the second or latter parameter. The type &quot;void&quot; means that the function takes no parameter, so it must be specified solely with no parameter name.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E4107C</td>
<td>invalid parameter declaration after void type in parameter list</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Though the first parameter in the parameter list is declared as the void type, another parameter follows. The &quot;void&quot; parameter means that the function takes no parameter.</td>
</tr>
<tr>
<td>E4108C</td>
<td>duplicate member <code>%I' in </code>%T'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The symbol indicated is already declared as a member of the struct or union.</td>
</tr>
<tr>
<td>E4109C</td>
<td>integral constant expression required for enumerator initialization of `%I'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Specify an integral constant as an initializer of the enumerator.</td>
</tr>
<tr>
<td>E4110C</td>
<td>undeclared identifier `%I'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The identifier used in the expression is not declared. An identifier must be declared before using it in an expression except for function call.</td>
</tr>
<tr>
<td>E4111C</td>
<td>bit-field size of `%D' is not integral constant expression</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The width of a bit-field must be specified by an integral constant expression.</td>
</tr>
<tr>
<td>E4112C</td>
<td>type of bit-field member `%D' is not integral type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>A bit-field member must be the integral type.</td>
</tr>
</tbody>
</table>
Appendix E  ERROR MESSAGE

E4113C  bit-field size %d of `%D' is too large

[Explanation]
The width of a bit-field must be equal to or less than that of its type.

E4114C  bit-field width of `%D' is negative

[Explanation]
The width of a bit-field must be zero or positive.

E4115C  bit-field width of `%D' must be non-zero

[Explanation]
The width of a bit-field must be positive.

E4116C  bit-field width %d of `%D' is larger than its type maximum width %d

[Explanation]
The width of a bit-field must be equal to or less than that of its type.

E4117C  size of `%T' is zero

[Explanation]
No member with non-zero width is declared in the struct or union tag. The size of the tag is equal to zero.

E4118C  %Z: invalid lvalue: string constant specified

[Explanation]
A character string constant cannot be an lvalue (an expression as a destination of the assignment operation).
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4119C</td>
<td>invalid operands: %s: %O</td>
<td>An invalid operand is specified. This message is built in E4001C, E4002C, E4005C, E4006C, E4007C, and E4008C.</td>
</tr>
<tr>
<td>E4120C</td>
<td>%Z: invalid lvalue</td>
<td>The specified expression cannot be an lvalue (an expression as a destination of the assignment operation).</td>
</tr>
<tr>
<td>E4121C</td>
<td>%Z: invalid lvalue: void type specified</td>
<td>An expression of the void type cannot be an lvalue (an expression as a destination of the assignment operation).</td>
</tr>
<tr>
<td>E4122C</td>
<td>%Z: invalid lvalue: function type specified</td>
<td>An expression of the function type (ex. a symbol of a function) cannot be an lvalue (an expression as a destination of the assignment operation).</td>
</tr>
<tr>
<td>E4123C</td>
<td>%Z: invalid lvalue: array type specified</td>
<td>An expression of the array type (ex. a symbol of an array) cannot be an lvalue (an expression as a destination of the assignment operation).</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E4124C</td>
<td>%Z: pointer type or array type required: %O</td>
<td>An expression of the pointer type or array type must be specified as an operand of the addition, subtraction or reference of an array operation where the pointer type is allowed.</td>
</tr>
<tr>
<td>E4125C</td>
<td>%Z: different struct/union types</td>
<td>In the assignment operation to the struct or union, the source and the destination must be both the struct or union of the same tag.</td>
</tr>
<tr>
<td>E4126C</td>
<td>void type function `%D' cannot return void</td>
<td>The return value of void type means that no value is returned as that of function, not that a value of the void type is returned. Do not specify any expression in a return statement.</td>
</tr>
<tr>
<td>E4127C</td>
<td>%Z: void type specified: `%D'</td>
<td>The void type is used in the context where the type is not allowed. For example, in an initialization.</td>
</tr>
<tr>
<td>E4128C</td>
<td>%Z: void type specified: argument %d</td>
<td>A void type expression is specified as an argument of a function call. The void type cannot be used in an expression.</td>
</tr>
<tr>
<td>E4129C</td>
<td>%Z: void type specified: %O</td>
<td>The void type is used in the context where the type is not allowed. For example, in an assignment.</td>
</tr>
</tbody>
</table>
E4130C  void type function `%D' cannot return value

[Explanation]
The return value of void means that no value is returned as that of function. Do not specify any expression in the return statement.

E4131C  %Z: extra argument: argument %d

[Explanation]
The parameter type of void means that a function takes no parameter. Do not specify any argument in a function call.

E4132C  %Z: invalid lvalue: void type specified: `%D'

[Explanation]
The target object of the initialization is the void type. It is not allowed as an lvalue (an expression as a destination of the assignment operation).

E4133C  %Z: invalid lvalue: void type specified: %O

[Explanation]
The destination of the assignment is the void type. It is not allowed as an lvalue (an expression as a destination of the assignment operation).

E4134C  %Z: invalid void type specified in function `%D'

[Explanation]
The expression of the void type cannot be specified in a return statement.

E4135C  %Z: %s: `%T': %O

[Explanation]
Cannot determine the size. This message is built in E4009C, E4010C, and E4014C.

[Note]
This message is not used.
### E4136C

%Z: `%s': `%T'

[Explanation]
Cannot determine the size. This message is built in E4009C, E4010C, and E4014C.

[Note]
This message is not used.

### E4137C

undefined field `%I': `%T' is incomplete type: %O

[Explanation]
Cannot find the member because no tag is defined for the struct or union.

### E4138C

field `%I' is undefined in `%T': %O

[Explanation]
Cannot find the member in the tag of the struct or union.

### E4139C

invalid type conversion from floating point type to pointer type

[Explanation]
It is impossible to convert from the floating point type to the pointer type because the resulting value is incorrect.

### E4140C

invalid type conversion from pointer type to floating point type

[Explanation]
It is impossible to convert from the pointer type to the floating point type because the resulting value is incorrect.

### E4141C

invalid type conversion: void value ought to be ignored

[Explanation]
The conversion from the void type to the struct or union type is invalid. The expression of the void type means that the resulting value of the expression is ignored.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4142C</td>
<td>invalid type conversion to array type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>It is invalid to convert to the array type.</td>
</tr>
<tr>
<td>E4143C</td>
<td>invalid type conversion to different non scalar type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The conversion to a different type of a struct or union is invalid.</td>
</tr>
<tr>
<td>E4144C</td>
<td>invalid type conversion to <code>%T</code></td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Types cannot be converted to each other.</td>
</tr>
<tr>
<td>E4145C</td>
<td>invalid type conversion to array type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The conversion from the array type to the struct or union type is invalid.</td>
</tr>
<tr>
<td>E4146C</td>
<td>invalid type conversion to function type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The conversion from the function type to the struct or union type is invalid.</td>
</tr>
<tr>
<td>E4147C</td>
<td>invalid type for conditional expression: <code>%T</code> specified: scalar type expected: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The struct or union type and the void type cannot be specified as an expression in a condition. Only the scalar type (arithmetic type or pointer type) is allowed.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error Message</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E4148C</td>
<td>duplicate case value %X in switch clause</td>
</tr>
<tr>
<td>E4149C</td>
<td>duplicate 'default' in switch clause</td>
</tr>
<tr>
<td>E4150C</td>
<td>integral type required for switch conditional expression: <code>%T</code> specified</td>
</tr>
<tr>
<td>E4151C</td>
<td>void type expression is used as argument: argument %d</td>
</tr>
<tr>
<td>E4152C</td>
<td>too many arguments to function: argument %d expect %d</td>
</tr>
<tr>
<td>E4153C</td>
<td>too few arguments to function: argument %d expect %d</td>
</tr>
</tbody>
</table>

[Note]
This message is not used.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
<th>Explanation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4154C</td>
<td>cannot take address of bit-field `%D'</td>
<td>The expression to show a bit-field is specified as an operand of &amp; operator. The address of a bit-field is not available.</td>
<td></td>
</tr>
<tr>
<td>E4155C</td>
<td>argument passing to function `%N': argument %d</td>
<td>The argument is passed to the function.</td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4156C</td>
<td>integral type required for case expression: `%T' specified</td>
<td>Only an integral type can be specified in the case label as in the condition expression of the switch statement.</td>
<td></td>
</tr>
<tr>
<td>E4157C</td>
<td>integral constant expression required for case expression</td>
<td>Only the integral constant can be specified in the case label expression.</td>
<td></td>
</tr>
<tr>
<td>E4158C</td>
<td>incompatible types between <code>%T' and </code>%T'</td>
<td>Types are incompatible in the redeclaration of the symbol by the external reference or in the assignment operation of a struct.</td>
<td></td>
</tr>
<tr>
<td>E4159C</td>
<td>redefinition of <code>%s</code> `%D'</td>
<td>The symbol is doubly defined.</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix E  ERROR MESSAGE

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4160C</td>
<td>%Z: void type specified: argument %d of `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The expression for a void type argument is specified in the function call. The type is not allowed in an expression.</td>
</tr>
</tbody>
</table>

| E4161C     | %Z: extra argument: argument %d of `%D' |
|            | [Explanation] |
|            | The parameter type of void means that a function takes no parameter. Do not specify any argument in a function call. |

| E4162C     | %Z: expected `%T' actual `%T': argument %d of `%D' |
|            | [Explanation] |
|            | The type of the argument in the function call is incompatible with that of the parameter declared. Modify the argument to match with the parameter type. |

| E4163C     | invalid type conversion of void type: invalid use of void |
|            | [Explanation] |
|            | An invalid type conversion is found related to the void type. |

| E4164C     | invalid `long long' type specifier |
|            | [Explanation] |
|            | This system does not support the long long type. |

| E4165C     | cannot take address of void type expression: %O |
|            | [Explanation] |
|            | The operand of the unary & operator is the void type. The address of a void type object is not available. |
E4166C  take address of rvalue: %O

[Explanation]
The operand of the unary & operator must be a lvalue (an expression as a destination of the assignment operation).

E4167C  `long char' type specifier is forbidden

[Explanation]
The long char type which is a part of an old style is supported no more.

E4168C  type of `%D' is incomplete type `%T'

[Explanation]
The struct, union or enumeration type without a tag name is an incomplete type. The type cannot be used in an expression.

E4169C  `%T' is incomplete type

[Explanation]
The struct, union or enumeration type without a tag name is an incomplete type. The type cannot be used in an expression.

E4170C  invalid type conversion

[Explanation]
Attempted to convert the constant to a type other than the void, integer, floating point, nor pointer type. A constant can be converted only to these types.

E4171C  %Z: expected `%T' actual `%T' in function `%D'

[Explanation]
The type of the return value for the function and the type of the expression in the return statement are incompatible.
E4172C  %Z: expected `%T' actual `%T': argument %d

[Explanation]
The type of the argument in the function call is incompatible with that of the corresponding parameter in the declaration. Modify the argument to match with the parameter type.

E4173C  %Z from `%T' to `%T': `%D'

[Explanation]
Types of the initializer and the target are incompatible.

E4174C  %Z from `%T' to `%T': %O

[Explanation]
Types of both sides for the assignment operation are incompatible.

E4175C  too many arguments to function: argument %d expect 0

[Explanation]
Though the parameter of the function is declared as the void type, the function is called with arguments. The parameter type of void means that the function takes no parameter, so no argument is needed.
The information of the function is issued in E4214C or E4215C.

E4176C  invalid type combination in declaration with obsolete modified typedef type

[Explanation]
The combination of types is invalid.

[Note]
This message is not used.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4177C</td>
<td>too many same data types specified for declaration of `%s'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The same type specifiers appear in a declaration.</td>
</tr>
<tr>
<td>E4178C</td>
<td>invalid void type parameter before ellipsis in parameter list</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The ellipsis cannot be specified with void in the parameter declaration.</td>
</tr>
<tr>
<td>E4179C</td>
<td>different kind tag declaration of `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The tag name of the struct, union or enumeration is already declared as</td>
</tr>
<tr>
<td></td>
<td>a different kind tag.</td>
</tr>
<tr>
<td>E4180C</td>
<td>width of type is too wide: cannot use `%T' for bit-field: declaration</td>
</tr>
<tr>
<td></td>
<td>of `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The width of the bit-field exceeds the maximum value of this system,</td>
</tr>
<tr>
<td></td>
<td>then it can not be regarded as a bit-field declaration.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4181C</td>
<td>invalid null subscript of array: `%D'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The subscript of the array is not specified. Cannot determine the size</td>
</tr>
<tr>
<td></td>
<td>of the symbol.</td>
</tr>
<tr>
<td>E4182C</td>
<td><code>%Z: </code>%T' is incomplete type</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The type is incomplete (the struct, union or enumeration whose tag is</td>
</tr>
</tbody>
</table>
|          | not defined or the array with no subscript specified).
### E4183C invalid storage class specifier `%A' specified for `%D' at %S

**[Explanation]**
The storage class specifier is invalid in this scope.

### E4184C sorry, internal limitation: precision of bit-field type %d is larger than native machine register width %d

**[Explanation]**
The width of the bit-field exceeds the maximum value of this system, then it can not be regarded as a bit-field declaration.

**[Note]**
This message is not used.

### E4185C %Z: missing argument

**[Explanation]**
This is an invalid argument specification for the built-in function.

**[Note]**
This message is not used.

### E4186C %Z: too many arguments

**[Explanation]**
This is an invalid argument specification for the built-in function.

**[Note]**
This message is not used.

### E4187C %Z: invalid argument: pointer required

**[Explanation]**
This is an invalid argument specification for the built-in function.

**[Note]**
This message is not used.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Description</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4188C</td>
<td>invalid type conversion to function</td>
<td>Cannot convert to the function type.</td>
</tr>
<tr>
<td>E4189C</td>
<td>automatic symbol has zero or negative size</td>
<td>An automatic variable has zero or negative size because of an invalid declaration of a tag or a subscript of an array.</td>
</tr>
<tr>
<td>E4190C</td>
<td>invalid typedefed function type for function definition</td>
<td>A typedef name of the function type cannot be used in the function definition.</td>
</tr>
<tr>
<td>E4191C</td>
<td>cannot quote EOF</td>
<td>The EOF is found just after the backslash in a character string or character constant.</td>
</tr>
<tr>
<td>E4192C</td>
<td>invalid `\' in input</td>
<td>A backslash is found outside a character string or character constant. It is an invalid character in a source file.</td>
</tr>
<tr>
<td>E4193C</td>
<td>asm: invalid type <code>\%T' of symbol </code>%D'</td>
<td>A symbol not of the array, arithmetic nor pointer type is specified in the LOAD/STORE notation in the asm statement.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>E4194C</td>
<td>asm: array `%D' cannot be used for store operand</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] A symbol of the array type is specified in the STORE notation in the asm statement. A symbol of the array type has an address of an array and cannot be specified in a STORE notation.</td>
<td></td>
</tr>
<tr>
<td>E4195C</td>
<td>asm: too many operands</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] Too many LOAD/STORE notations are written in an asm statement. Though the maximum for the internal process of the compiler is 6, whether or not they are valid actually depends on the kind of the asm directive.</td>
<td></td>
</tr>
<tr>
<td>E4196C</td>
<td>asm: invalid expression: `%!l'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] An invalid symbol is specified in the LOAD/STORE notation in the asm statement.</td>
<td></td>
</tr>
<tr>
<td>E4197C</td>
<td>asm: regular string constant expected for asm statement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The operand of the asm statement must be a simple string literal.</td>
<td></td>
</tr>
<tr>
<td>E4198C</td>
<td>asm: identifier is expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] No identifier is specified in the LOAD/STORE notation in the asm statement.</td>
<td></td>
</tr>
<tr>
<td>E4199C</td>
<td>asm: `%c' is expected</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The identifier is not enclosed with <code>}' nor </code>]' in the LOAD/STORE notation in the asm statement.</td>
<td></td>
</tr>
</tbody>
</table>
### APPENDIX

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4200C</td>
<td><code>asm: typedefed identifier </code>%l' cannot be used in asm statement`</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The identifier specified in the LOAD/STORE notation in the asm statement is a typedef name.</td>
</tr>
<tr>
<td>E4201C</td>
<td><code>asm: invalid assignment of enumerator </code>%l'`</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The identifier specified in the STORE notation in the asm statement is an enumeration constant. Cannot &quot;STORE&quot; to an enumeration constant.</td>
</tr>
<tr>
<td>E4202C</td>
<td><code>asm: invalid expression</code></td>
</tr>
<tr>
<td></td>
<td>[Explanation] An invalid string literal is found in the asm statement.</td>
</tr>
<tr>
<td>E4203C</td>
<td><code>asm: invalid use of enumerator </code>%l'`</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The identifier specified in the LOAD notation in the asm statement is an enumeration constant. Cannot &quot;LOAD&quot; an enumeration constant.</td>
</tr>
<tr>
<td>E4204C</td>
<td><code>default' outside switch</code></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The default label is written outside the switch statement. A default label is a default branch in the case no case label in the switch statement matches.</td>
</tr>
<tr>
<td>E4205C</td>
<td><code>sorry, internal limitation: number of array dimensions exceeds %d</code></td>
</tr>
<tr>
<td></td>
<td>[Explanation] The dimension of an array exceeds the internal maximum value of the compiler.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>E4206C</td>
<td>%Z: cannot get alignment of `%T'</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Cannot get boundary alignment of an incomplete type.</td>
</tr>
<tr>
<td></td>
<td>[Note]</td>
</tr>
<tr>
<td></td>
<td>This message is not used.</td>
</tr>
<tr>
<td>E4207C</td>
<td>invalid type combination of <code>%T' and </code>%T': %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The combination of types for operands is invalid.</td>
</tr>
<tr>
<td>E4208C</td>
<td>invalid type combination of <code>%T' and </code>%T': integral type required: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The combination of types for operands is invalid. An operand of the integral type is needed.</td>
</tr>
<tr>
<td>E4209C</td>
<td>invalid type combination of <code>%T' and </code>%T': arithmetic type required: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The combination of types for operands is invalid. An operand of the arithmetic type is needed.</td>
</tr>
<tr>
<td>E4210C</td>
<td>invalid type combination of <code>%T' and </code>%T': void type required: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The combination of types for operands is invalid. An operand of the void type is needed.</td>
</tr>
<tr>
<td>E4211C</td>
<td>invalid type combination of <code>%T' and </code>%T': compatible types required: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The combination of types for operands is invalid. An operand of a compatible type is needed.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>E4212C</td>
<td>only one argument is specified</td>
</tr>
<tr>
<td>E4213C</td>
<td>too many argument specified</td>
</tr>
<tr>
<td>E4214C</td>
<td>argument passing to builtin function <code>\%N</code></td>
</tr>
<tr>
<td>E4215C</td>
<td>argument passing to function <code>\%N</code>: &quot;%s&quot;, line %</td>
</tr>
<tr>
<td>E4216C</td>
<td>invalid declaration specifier: <code>\%s</code></td>
</tr>
<tr>
<td>E4217C</td>
<td>member <code>\%D</code> has invalid storage class specifier</td>
</tr>
</tbody>
</table>

[Note]
This message is not used.
Appendix E  ERROR MESSAGE

E4218C  cannot get unknown scaling value

[Explanation]
The operation is invalid because the size of the expression is unknown.

E4219C  `case' outside switch

[Explanation]
The case label is written outside the switch statement. A case label is a branch in the switch statement.

E4220C  invalid data member: `)%D'

[Explanation]
An invalid type is specified to the member of the struct or union.

E4221C  invalid `...' in expression

[Explanation]
The ellipsis `...' can be used only in the parameter description position.

E4222C  invalid `$' in input

[Explanation]
The `$' is not included in the source character set.

E4223C  specification of sign of plain `char' conflict

[Explanation]
Do not specify following two options simultaneously, one specifies that the simple char is signed and the other unsigned.
### APPENDIX

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4224C</td>
<td>#pragma intvect: the same vector number is expected</td>
</tr>
</tbody>
</table>

**[Explanation]**

Another function is already declared by "#pragma intvect" with the same vector number.

| E4225C | cannot take address of object in register storage |

**[Explanation]**

A symbol of an operand of & operator is declared with the "register" storage class specifier. The address of an object for that kind is not available.

| E4226C | #pragma register: syntax error: register bank number is expected |

**[Explanation]**

A register bank number is needed for "#pragma register" directive.

| E4227C | #pragma register: register bank number is out of range |

**[Explanation]**

A register bank number of the "#pragma register" directive must be equal to or less than 31.

| E4228C | #pragma noregister: `#pragma register' not exist |

**[Explanation]**

The preceding "#pragma register" is missing for the "#pragma noregister".

| E4229C | #pragma ilm: syntax error: interrupt level is expected |

**[Explanation]**

Interrupt level is needed for "#pragma ilm" directive.
Appendix E  ERROR MESSAGE

E4230C  #pragma ilm: interrupt level is out of range

[Explanation]
Interrupt level of "#pragma ilm" must be equal to or less than 7.

E4231C  #pragma noilm: `#pragma ilm' not exist

[Explanation]
The preceding "#pragma ilm" is missing for the "#pragma noilm".

E4232C  #pragma except: `#pragma ssb' exist

[Explanation]
The "#pragma except" cannot be between "#pragma ssb" and "#pragma nossb".

E4233C  #pragma noexcept: `#pragma except' not exist

[Explanation]
The preceding "#pragma except" is missing for the "#pragma noexcept".

E4234C  #pragma ssb: `#pragma except' exist

[Explanation]
The "#pragma ssb" cannot be between "#pragma except" and "#pragma noexcept".

E4235C  #pragma nossb: `#pragma ssb' not exist

[Explanation]
The preceding "#pragma ssb" is missing for the "#pragma nossb".
### APPENDIX

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4236C</td>
<td>#pragma register: invalid constant for register bank number</td>
</tr>
<tr>
<td></td>
<td>[Explanation] An invalid integral constant is specified as a register</td>
</tr>
<tr>
<td></td>
<td>number in the &quot;#pragma register&quot;.</td>
</tr>
<tr>
<td>E4237C</td>
<td>#pragma ilm: invalid constant for interrupt level value</td>
</tr>
<tr>
<td></td>
<td>[Explanation] An invalid integral constant is specified as an interrupt</td>
</tr>
<tr>
<td></td>
<td>level in the &quot;#pragma ilm&quot;.</td>
</tr>
<tr>
<td>E4238C</td>
<td>pointer to struct/union type required for left operand: access to</td>
</tr>
<tr>
<td></td>
<td>member <code>%I</code>: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A type other than the pointer type to a struct or union</td>
</tr>
<tr>
<td></td>
<td>is specified on the left-hand side of the &quot;-%&quot; operator.</td>
</tr>
<tr>
<td>E4239C</td>
<td>struct/union type required for left operand: access to member <code>%I</code>: %O</td>
</tr>
<tr>
<td></td>
<td>[Explanation] A type other than struct or union is specified on the</td>
</tr>
<tr>
<td></td>
<td>left-hand side of the operator.</td>
</tr>
<tr>
<td>E4240C</td>
<td>#pragma ilm: interrupt level number should be an integer value without</td>
</tr>
<tr>
<td></td>
<td>the sign</td>
</tr>
<tr>
<td></td>
<td>[Explanation] An interrupt level number is specified badly for the</td>
</tr>
<tr>
<td></td>
<td>&quot;#pragma ilm&quot;. It must be an integer value with no sign specified.</td>
</tr>
<tr>
<td>E4241C</td>
<td>invalid restrict qualifier</td>
</tr>
<tr>
<td></td>
<td>[Explanation] The type qualifier &quot;restrict&quot; is effective only on an</td>
</tr>
<tr>
<td></td>
<td>array or a pointer type.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Message Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>E4242C</td>
<td>invalid type qualifier in array</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The type qualifier which is applicable to a subscript of an array is the &quot;restrict&quot; only.</td>
</tr>
<tr>
<td>E4243C</td>
<td>sorry, internal limitation: stack size exceeds 32768 bytes</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The stack size cannot exceed 32768 bytes in a function.</td>
</tr>
<tr>
<td>E4244C</td>
<td>array is too large</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>The array size cannot be expressed by the size_t type.</td>
</tr>
<tr>
<td>F9001C</td>
<td>too many errors</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Too many errors are detected to continue.</td>
</tr>
<tr>
<td>F9002C</td>
<td>detected too many error to terminate compilation</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>Too many syntax errors are detected to continue.</td>
</tr>
<tr>
<td>F9003C</td>
<td>too large message file</td>
</tr>
<tr>
<td></td>
<td>[Explanation]</td>
</tr>
<tr>
<td></td>
<td>This is invalid message file.</td>
</tr>
<tr>
<td>Code</td>
<td>Error Message</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>F9004C</td>
<td>too many messages</td>
</tr>
<tr>
<td>F9005C</td>
<td>illegal message file</td>
</tr>
<tr>
<td>F9006C</td>
<td>virtual memory exhausted</td>
</tr>
</tbody>
</table>
INDEX

Symbols
__div built-in function ........................................... 113
__divu built-in function ......................................... 114
__mod built-in function......................................... 113
__modu built-in function....................................... 115
__mul built-in function .......................................... 112
__mulu built-in function ........................................ 114
__wait_nop built-in function ................................. 111
_abort function ..................................................... 145
_exit function........................................................ 144

Numerics
16-bit/24-bit addressing access function.............. 100

A
access function ........................................ 97, 98, 100
ANSI standard...................................................... 116
area management................................................ 156
asm statement .................................................... 92
assert.h ................................................................ 164
automatic distinction function............................... 109

B
built-in function..................... 111, 112, 113, 114, 115

C
C COMPILER........................................................... 2
case sensitiveness................................................. 17
close function .............................................. 138, 161
command cancel option ........................................ 20
command line.................................................. 14, 17
command operand ................................................... 15
command option ..................................................... 18
command related option ....................................... 22, 50
comment entry in option file................................. 53
compiler ................................................................. 63
compiler translation.............................................. 117
compiler-dependent language specification differential ........................................ 148
ctype.h ................................................................. 164

dependency ............................................................17
description by asm statement.................................92
description by pragma instruction...........................93
direct area access function .................................... 98
directory name ..................................................... 16

e
error level................................................................54
e-error output file .................................................. 133
e-example ................................................................... 159
exclusiveness ......................................................... 17
execution process..................................................120
expressible value range........................................ 150
extension format ..................................................... 73, 82

F
fcc907s command ............................................3, 116
fcntl.h ....................................................................166
FELANG .................................................................12
FETOOL ...................................................................8
file name .................................................................16
file system .............................................................. 156
file type ................................................................. 126
float.h....................................................................164
floating-point data format.................................150, 151, 152
function .................................................................132
function description function ...................................96

G
generation rule for name used by compiler ............63

H
header file ................................................................5

I
I/O area access function ....................................... 97
I/O port ................................................................. 160
INC907 .................................................................10
initialization ....................................................... 133, 157
in-line expansion specifying function ....................102
input .........................................................................133
interrupt func. function ..........................................110
interrupt function description function .................96
interrupt level setup function ....................................96
interrupt mask disable function ...............................95
interrupt mask setup function ...................................95
interrupt stack frame ...............................................88
interrupt vector table generation function ...............96
INDEX

isatty function .................................................. 142, 161

L
language specification ................................. 148
language specification related option ........... 22, 33
level setup function ....................................... 95
library .......................................................... 167
library function ........................................... 135
library section name .................................... 128
library use .................................................... 132
limitation on compiler translation .............. 117
limits.h ......................................................... 164
list of command cancel option ..................... 20
list of command option ................................ 18
load module creation .................................... 157
low-level function ......................................... 135, 136
low-level function (system-dependent process) type .................................................. 129
low-level function library ............................. 156
low-level function type .................................. 134
lseek function ............................................... 141, 161

M
macro predefined by fcc907s command .......... 116
macro stipulated by ANSI standard .............. 116
mask disable function ................................... 95
mask setup function ........................................ 95
math.h ......................................................... 164, 165
memory model ............................................... 61
message generated in translation process .......... 54
module creation ............................................. 157
multiple specifying of same option ................. 17

N
name used by compiler ................................... 63
no-register-save interrupt func. function ...... 110

O
open function ................................................ 137, 160
operation specific to library ......................... 167, 173
OPT907.......................................................... 9
optimization ................................................. 6
optimization related option ......................... 22, 37
option file .................................................... 52, 53
option file general format .................................. 52
option file limitation .................................... 52
option file related option ......................... 22, 51
option syntax ................................................ 17
options for compiling process control .......... 3
output .......................................................... 133
output file .................................................... 133
output object related option ....................... 22, 44

P
position within command line ....................... 17
pragma instruction ......................................... 93
preprocessor related option ......................... 22, 25
process ......................................................... 132
process/low-level function ......................... 135

R
read function ............................................... 139, 161
register bank number setup function .......... 106

S
sbrk function ................................................. 143, 161
section name change function ...................... 103
sensitiveness ............................................... 109
simulator debugger setup ............................ 159
special I/O port ............................................ 160
specifying function ........................................ 102, 108
stack bank automatic distinction function .... 109
standard error output file ......................... 133, 133
standard input/output .................................. 133
standard library function ............................ 135
stdarg.h ....................................................... 165
stddef.h ...................................................... 165
stdio.h ......................................................... 165
stdlib.h ....................................................... 165
strcat.h ....................................................... 166
stream area initialization ............................ 133
string.h ....................................................... 166
symbolic debugger ......................................... 6
syntax ........................................................... 17
sys/types.h ................................................... 166

T
termination process ....................................... 133
TMP ......................................................... 11
tool identifier .............................................. 54
translation control related option ............... 22, 23

U
union ........................................................... 67
unistd.h ....................................................... 166
INDEX

V
vector table generation function............................ 96

W
write function................................................ 140, 161