

F²MC-8L/8FX FAMILY
8-BIT MICROCONTROLLER
SOFTUNE™ C COMPILER MANUAL

Support Soft Manual



F²MC-8L/8FX FAMILY
8-BIT MICROCONTROLLER
SOFTUNE™ C COMPILER MANUAL

Support Soft Manual



PREFACE

■ Objective of This Manual and Target Readers

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

This manual is prepared for persons who use the above-mentioned compiler and to create and develop application programs in C language. Be sure to read this manual completely.

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 as "ANSI standard" in this manual.

■ Trademarks

SOFTUNE is a trademark of Spansion LLC.

Windows is registered trademarks of Microsoft Corporation 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.

The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

■ Composition of Manual

This manual consists of the following 10 chapters and appendix.

CHAPTER 1 SOFTUNE C COMPILER OVERVIEW

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.

(for the setting of environment variables, refer to the manual for each operating system).

CHAPTER 3 C COMPILER 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 standards based on the ANSI standards.

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.

■ 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)

CONTENTS

CHAPTER 1	SOFTUNE C COMPILER OVERVIEW	1
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	7
2.1	FETOOL	8
2.2	LIB896	9
2.3	OPT896	10
2.4	INC896	11
2.5	TMP	12
2.6	FELANG	13
CHAPTER 3	C COMPILER OPERATION	15
3.1	Command Line	16
3.2	Command Operands	17
3.3	File Names and Directory Names	18
3.4	Command Options	19
3.4.1	List of Command Options	20
3.4.2	List of Command Cancel Options	23
3.5	Details of Options	25
3.5.1	Translation Control Related Options	26
3.5.2	Preprocessor Related Options	28
3.5.3	Data Output Related Options	31
3.5.4	Language Specification Related Options	36
3.5.5	Optimization Related Options	41
3.5.6	Output Object Related Options	46
3.5.7	Debug Information Related Options	47
3.5.8	Command Related Options	48
3.5.9	Linkage Related Options	49
3.5.10	Option File Related Options	51
3.6	Option Files	52
3.7	Messages Generated in Translation Process	54
CHAPTER 4	OBJECT PROGRAM STRUCTURE	57
4.1	Section Structure of fcc896s Command	58
4.2	Generation Rules for Names Used by Compiler	60
4.3	Boundary Alignment of fcc896s Command	61
4.4	Bit Field of fcc896s Command	62
4.5	Structure/Union of fcc896s Command	64
4.6	Function Call Interface of fcc896s Command	65
4.6.1	Stack Frame of fcc896s Command	66
4.6.2	Argument of fcc896s Command	67
4.6.3	Argument Extension Format of fcc896s Command	68
4.6.4	Calling Procedure of fcc896s Command	69
4.6.5	Register of fcc896s Command	70

4.6.6	Return Value of fcc896s Command	71
4.7	Interrupt Function Call Interface of fcc896s Command	72
4.7.1	Interrupt Stack Frame of fcc896s Command	73
4.7.2	Interrupt Function Calling Procedure of fcc896s Command	74
CHAPTER 5	EXTENDED LANGUAGE SPECIFICATIONS	75
5.1	Assembler Description Functions	76
5.2	Interrupt Control Functions	79
5.3	I/O Area Access Function	82
5.4	direct Area Access Function	83
5.5	In-line Expansion Specifying Function	84
5.6	Section Name Change Function	85
5.7	Register Bank Number Setup Function	88
5.8	Interrupt Level Setup Function	89
5.9	No-Register-Save Interrupt Func. Function	90
5.10	Built-in Function	91
5.11	Predefined Macros	92
5.12	Limitations on Compiler Translation	93
CHAPTER 6	EXECUTION ENVIRONMENT	95
6.1	Execution Process Overview	96
6.2	Startup Routine Creation	98
CHAPTER 7	LIBRARY OVERVIEW	99
7.1	File Organization	100
7.2	Relationship to Library Incorporating System	102
CHAPTER 8	LIBRARY INCORPORATION	103
8.1	Library Incorporation Overview	104
8.2	Low-level Function Types	105
8.3	Standard Library Functions and Required Process/Low-Level Functions	106
8.4	Low-Level Function Specifications	107
8.4.1	sbrk Function	108
8.4.2	_exit Function	109
8.4.3	_abort Function	110
CHAPTER 9	COMPILER-DEPENDENT SPECIFICATIONS	111
9.1	Compiler-Dependent Language Specification Differentials	112
9.2	Floating-Point Data Format and Expressible Value Range	114
9.3	Floating-Point Data Operation due to the Runtime Library Function	115
CHAPTER 10	SIMULATOR DEBUGGER LOW-LEVEL FUNCTION LIBRARY	119
10.1	Low-Level Function Library Overview	120
10.2	fcc896s Command Low-Level Function Library Use	121
10.3	Low-Level Func. Function	122
10.4	Low-Level Function Library Change	123

APPENDIX	125
APPENDIX A List of Type, Macro, Variable, and Function	126
APPENDIX B Operations Specific to Libraries	129
APPENDIX C ANSI Standard Non-Conforming Specifications	131
APPENDIX D About Reentrancy of the C Library Functions	134
APPENDIX E Error Message	136
APPENDIX F Major Changes	277
 INDEX.....	 279

CHAPTER 1

SOFTUNE C COMPILER

OVERVIEW

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 Spansion-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 represented in 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

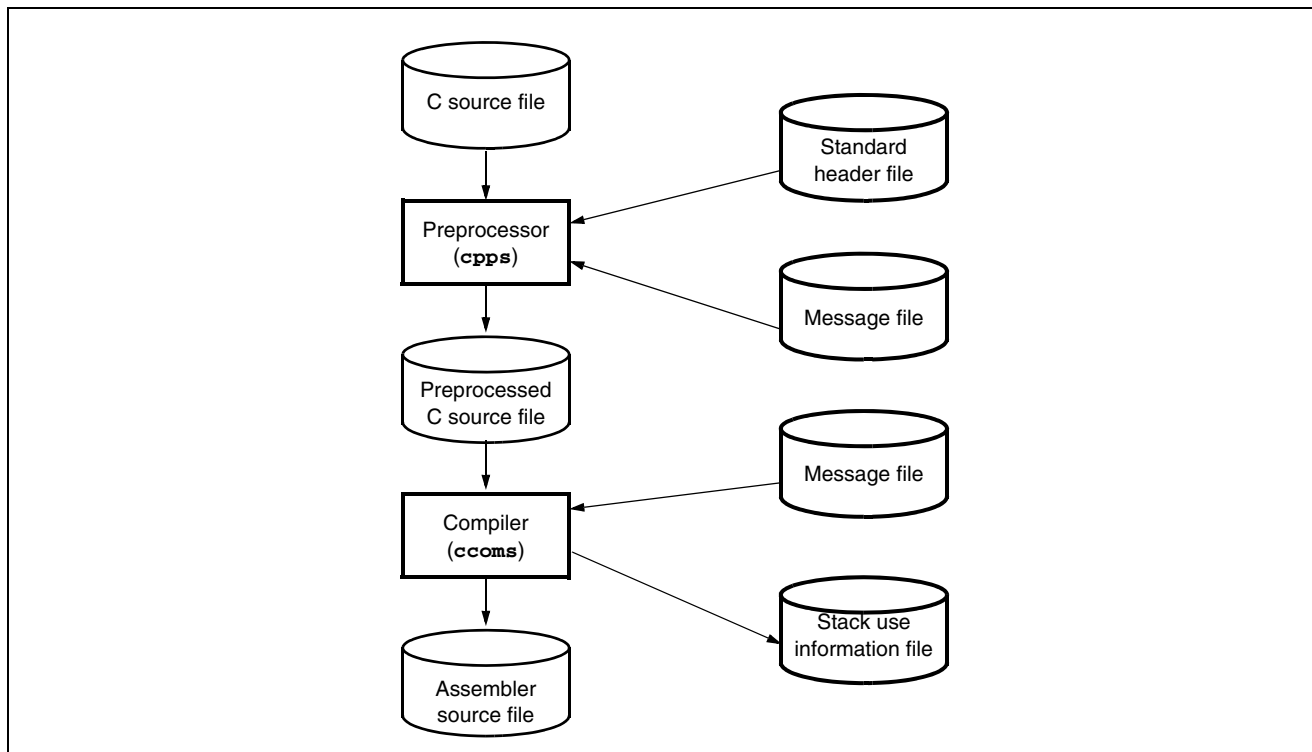
Preprocessing is conducted by the preprocessor (cpps) which is a subcomponent of the compiler. Preprocessing instructions (#if, #define, #include, etc.) in a C source file are interpreted and converted to a preprocessed C source file.

● 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 fcc896s command is to be used. These commands automatically call 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.

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 C COMPILER OPERATION".

1.2 Basic Process of Commands

The basic process of commands to be used in the C compiler is described below.
The C compiler has the following commands.

- **fcc896s:** For F²MC-8L/8FX family command

■ fcc896s Command Basic Process

The fcc896s command basically generates an absolute 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 fcc896s command is given below, where > is the command prompt.

[Example 1]

```
> fcc896s -cpu MB89P935B 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 absolute file (file.abs) in the current directory.

[Example 2]

```
> fcc896s -o outfile -cpu MB89P935B file.c
```

At the input given above, the command generates an absolute 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.

● -c option

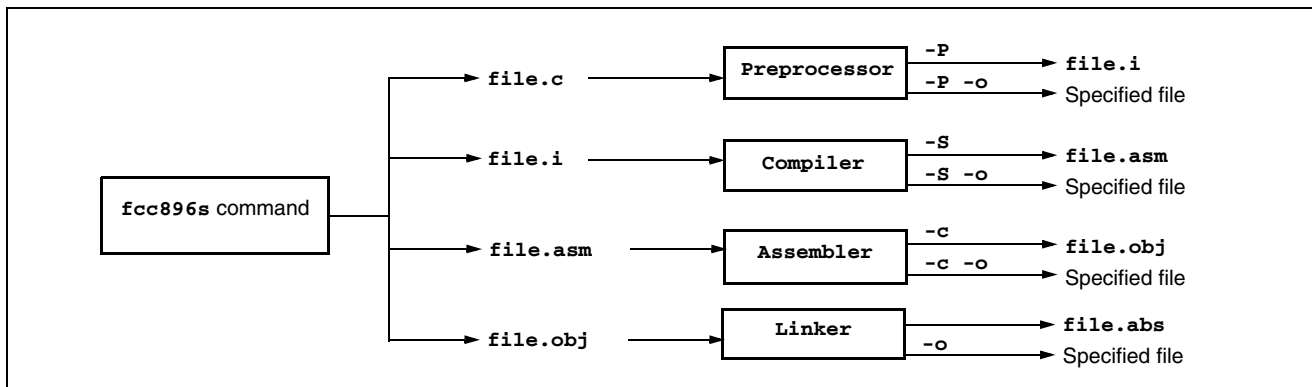
When the -c option is specified, the command calls up the preprocessor, compiler, and assembler and performs preprocessing, compiling, and assembling to generate an object file in the current directory. The extension of the generated file is changed to .obj.

● -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 fcc896s 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 basic 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 angle brackets (`<>`) is searched for 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 a directory specified by the absolute path name. If the absolute path name is not specified, such a header file is searched for 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]

```
> fcc896s -cpu MB89P935B -I ..\include file.c
```

At the input given above, the command searches for the header file enclosed within angle 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.

■ 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 symbolic debuggers are available; simulator debugger and emulator debugger.

When the optimization option (`-O [1-4]`) is specified, notes the following points to execute the debugging.

When the optimization option 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 is 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.

■ Optimization

When the -O option is specified, the compiler generates an object subjected to general-purpose optimization.

CHAPTER 2

SETUP OF SYSTEM ENVIRONMENT BEFORE USING C COMPILER

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:\softune

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

- 2.1 FETOOL
- 2.2 LIB896
- 2.3 OPT896
- 2.4 INC896
- 2.5 TMP
- 2.6 FELANG

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 LIB896

Specify the directory that contains the library to which the fcc896s command is linked by default.

■ LIB896

[General Format 1] For UNIX OS

```
setenv LIB896 Library directory [: Directory 2 ...]
```

[General Format 2] For Windows

```
set LIB896=Library directory [; Directory 2 ...]
```

Specify the directory of library to which linking is effected by default.

If LIB896 setup is not completed, the directory placed at a relativity from the directory specified by FETOOL (%FETOOL%\lib\896) is regarded as the default library directory.

When two or more directories are specified, ":" (UNIX) or ";" (Windows) is interpreted as the directory name delimiter.

[Example] For UNIX OS

```
setenv LIB896 /usr/local/softune/lib/896
```

[Example] For Windows

```
set LIB896=d:\softune\lib\896
```

2.3 OPT896

Specify the directory for the default option file to be used by the fcc896s command.

■ OPT896

[General Format 1] For UNIX OS

setenv OPT896 Default option file directory

[General Format 2] For Windows

set OPT896=Default option file directory

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

If OPT896 setup is not completed, the directory placed at a relativity from the directory specified by FETOOL (%FETOOL%\lib\896) is regarded as the default option file directory.

No more than one directory can be specified.

[Example] For UNIX OS

```
setenv OPT896 /usr/local/softune/lib/896
```

[Example] For Windows

```
set OPT896=c:\softune\lib\896
```

2.4 INC896

Specify the directory where a standard header file search is to be conducted by the fcc896s command.

■ INC896

[General Format 1] For UNIX OS

setenv INC896 Standard include directory

[General Format 2] For Windows

set INC896=Standard include directory

Specify the directory where the standard header file is to be searched for. The directory specified by INC896 is regarded as the standard include directory.

If INC896 setup is not completed, the directory placed at a relativity from the directory specified by FETOOL (%FETOOL%\lib\896\include) is regarded as the standard header file directory.

No more than one directory can be specified.

[Example] For UNIX OS

```
setenv INC896 /usr/local/softune/lib/896/include
```

[Example] For Windows

```
set INC896=c:\softune\lib\896\include
```

2.5 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
```

2.6 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

C COMPILER 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

3.1 Command Line

The command line format is shown below.

- **fcc896s [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]

```
>fcc896s file1.c -S -I \home\myincs file2.c -cpu MB89P935B
```

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]

```
>fcc896s file1.c file2.c file3.c -cpu MB89P935B
```

Files named file1.c, file2.c, and file3.c are subjected to preprocessing, compiling, assembling, and linking so that files named file1.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

Extension	Command Process
.c	The file having this extension is regarded as a C source file and subjected to preprocessing and subsequent processes.
.i	The file having this extension is regarded as a preprocessed C source file and subjected to compiling and subsequent processes.
.asm	The file having this extension is regarded as a compiled assembler source file and subjected to assembling and subsequent processes.
.obj	The file having this extension is regarded as an assembled object file and subjected to linking and subsequent processes.
.abs	The file having this extension is regarded as a linked absolute file, and an error output is generated. No absolute file can be specified.

Note:

The associated process may be inhibited depending on the option specifying.

[Example]

```
>fcc896s file1.c file2.i -cpu MB89P935B
```

A file named file1.c is subjected to preprocessing, compiling, and assembling. A file named file2.i is then subjected to compiling, assembling, and linking, in order named, to generate a file named file1.abs.

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]

```
>fcc896s -o outfile file.c -o outobj -cpu MB89P935B
```

The resultant output file name will be outobj.

The options listed below can be specified more than once for the same command and are different in meaning each time.

- Options that are significant when specified more than one time

```
-D -f -I -INF -K -L -l -ra -ro -sc -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) >fcc896s -C -E file1.c file2.c -cpu MB89P935B
```

```
2) >fcc896s file1.c -E file2.c -C -cpu MB89P935B
```

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 Sensitiveness

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 (1 / 2)

Specifying Format	Function
-B	Allows the C++ style comments(//)
-C	Leaves a comment in the preprocessing result
-cmsg	Outputs the compiling process end message to the standard output
-cpu MB number	Specifies the MB number of the CPU to be used
-cwno	Sets end code to 1 when warning given
-D name[=[tokens]]	Defines the macro name
-E	Performs preprocessing only and outputs the result to the standard output
-f filename	Specifies the option file
-g	Adds the information necessary for debugging to the object
-H	Outputs the acquired header file path name to the standard output
-help	Outputs the option list to the standard output
-I dir	Specifies the directory for header file search
-INF LIST	Generates the assemble list
-INF {SRCIN LINENO}	Inserts the associated C source information as a comment into the assembler source
-INF STACK[=filename]	Generates the stack use amount data
-J {a c n}	Specifies the specification level of the language to be interpreted by the compiler
-K {DCONST FCONST}	Specifies the type of a real constant without a suffix
-K EOPT	Effects optimization for changing the arithmetic operation evaluation procedure
-K LIB	Recognizes the standard function operation and implements in-line expansion/substitution for other functions
-K NOALIAS	Effects optimization on the presumption that differing pointers do not indicate the same area
-K NOINTLIB	Effects no in-line expansion for interrupt related functions
-K NOUNROLL	Inhibits loop unrolling

Table 3.4-1 List of Command Options (2 / 2)

Specifying Format	Function
-K NOVOLATILE	Does not consider __io qualifier variables to be volatile
-K REALOS	Effects in-line expansion for the ITRON system call function
-K {SIZE SPEED}	Selects optimization with emphasis placed on the size and execution speed
-K {UCHAR SCHAR}	Specifies the mere char sign handling
-K {UBIT SBIT}	Specifies the mere int bit field sign handling
-kanji {SJIS EUC}	Specifies kanji code used in program
-O level	Gives instructions for general-purpose optimization
-o path name	Outputs the result to the path name
-P	Performs preprocessing only and outputs the result to .i
-S	Performs processes up to compiling and outputs the result to .asm
-s defname=newname [, attr [, address]]	Changes the section name
-T item, arg1 [, arg2 ...]	Passes arguments to the tool
-U name	Cancels the macro name definition
-V	Outputs the executed compiler tool version information to the standard output
-w level	Specifies the warning message output level
-Xdof	Inhibits the default option file read operation
-x func [, func2 ...]	Specifies the in-line expansion of functions
-xauto [size]	Specifies the in-line expansion of the functions whose logical line count is not less than size
-Y item, dir	Changes the item position to dir

Table 3.4-2 List of fcc896s Command Options

Specifying Format	Function
-c	Performs processes up to assembling and outputs the result to .obj
-e name	Specifies the entry of a program
-L path1 [, path2 ...]	Specifies the library path
-l lib1 [, lib2 ...]	Specifies the library file name
-K ADDSP	Releases argument areas altogether
-K ARRAY	Optimization of array element access code.
-K NO_REDUCED_TYPE_OPERATION	The following ANSI standard non-conforming specifications are invalidated. • The integral promotions are not applied. • The type of the integer constant has a minimum type. • The type of the enumeration type has a minimum type. • The type of the truth-value of the operation result has type char.
-K NO_REDUCED_TYPE_ARGUMENT	The following ANSI standard non-conforming specification is invalidated. • The default argument promotions are not applied.
-m	Outputs a map file at the time of linking
-ra name = start/end	Specifies the RAM area
-ro name = start/end	Specifies the ROM area
-sc param	Specifies the section arrangement
-startup file	Specifies the startup file name

3.4.2 List of Command Cancel Options

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

■ List of Command Cancel Options

Table 3.4-3 List of Command Cancel Options (1 / 2)

Specifying Format	Function
-XB	Cancels the -B option
-XC	Cancels the -C option
-Xcmsg	Cancels the -cmsg option
-Xcwno	Cancels the -cwno option
-Xf	Cancels the -f option
-Xg	Cancels the -g option
-XH	Cancels the -H option
-Xhelp	Cancels the -help option
-XI	Cancels the -I option
-INF NOLINENO	Cancels the LINENO suboption
-INF NOLIST	Cancels the LIST suboption
-INF NOSRCIN	Cancels the SRCIN suboption
-INF NOSTACK	Cancels the STACK suboption
-K ALIAS	Cancels the NOALIAS suboption
-K INTLIB	Cancels the NOINTLIB suboption
-K NOEOPT	Cancels the EOPT suboption
-K NOLIB	Cancels the LIB suboption
-K NOREALOS	Cancels the REALOS suboption
-K UNROLL	Cancels the NOUNROLL suboption
-K VOLATILE	Cancels the NOVOLATILE suboption
-Xo	Cancels the -o option
-Xs	Cancels the -s option
-XT item	Cancels the -T item specifying
-XV	Cancels the -V option

Table 3.4-3 List of Command Cancel Options (2 / 2)

Specifying Format	Function
-Xx	Cancels the -x option
-Xxauto	Cancels the -xauto option
-XY item	Cancels the -Y item specifying

Table 3.4-4 List of fcc896s Command Cancel Options

Specifying Format	Function
-Xe	Cancels the -e option
-K NOADDSP	Cancels the ADDSP suboption
-K NOARRAY	Cancels the ARRAY suboption
-K REDUCED_TYPE_OPERATION	Cancels the NO_REDUCED_TYPE_OPERATION suboption
-K REDUCED_TYPE_ARGUMENT	Cancels the NO_REDUCED_TYPE_ARGUMENT suboption
-XL	Cancels the -L option
-Xl	Cancels the -l option
-Xm	Cancels the -m option
-Xra	Cancels the -ra option
-Xro	Cancels the -ro option
-Xsc	Cancels the -sc option
-Xstartup	Cancels the -startup option

3.5 Details of Options

This section details the options.

■ Translation Control Related Options

The translation control related options are related to preprocessor, compiler, assembler, and linker 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 called by commands.

■ Linkage Related Options

The linkage related options are related to linkage.

■ Option File Related Options

The option file related options are related to option files.

3.5.1 Translation Control Related Options

This section describes the options related to preprocessor, compiler, assembler, and linker 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 > -c

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

Table 3.5-1 Translation Control Related Option Exclusiveness

Specified Option	Option Invalidated
-E	-S and -c
-P	-S and -c
-S	-c
-c	None

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

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]

```
>fcc896s -E -cpu MB89P935B 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]

```
>fcc896s -P -cpu MB89P935B sample.c
```

The sample.c preprocessing result is output to the sample.i.

■ -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]

```
>fcc896s -S -cpu MB89P935B sample.c
```

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

■ -c

This option performs processes up to assembling and outputs the resultant object to file extension changed to .obj. If the input file is an object file, the -c option does not do anything.

[Example]

```
>fcc896s -c -cpu MB89P935B sample.c
```

The sample.c preprocessing and compiling process and assembling process results are output to the sample.obj.

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

Table 3.5-2 Relationship between File Types and Processes for Translation Control Related Options

Option File Type (Extension)	-E	-P	-S	-c	Nothing Specified
C source file (.c)	P	P	P and C	P, C and A	P, C, A and L
Preprocessed C source file (.i)	—	—	C	C and A	C, A and L
Assembler source file (.asm)	—	—	—	A	A and L
Object file (.obj)	—	—	—	—	L

P : Preprocessing

C: Compiling

A: Assembling

L : Linking

[Example]

```
>fcc896s -E file1.c file2.i -cpu MB89P935B
```

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.

```
>fcc896s -E file1.c file2.i file3.asm -cpu MB89P935B
```

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.

[Usage 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:

```
fcc896s -C -E -cpu MB89P935B 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]

```
>fcc896s -D os=m -D sys file.c -cpu MB89P935B
```

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 path names acquired during preprocessing to the standard output. The path names are sequentially output, one for each line, in the order of acquisition. If there are any two exactly the same path names, only the first one will be output. 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 output.

The -XH option cancels the -H option.

[Output Example]

• Input:

```
#include <stdio.h>
#include "head.h"
```

• Operation:

```
fcc896s -H -cpu MB89P935B sample.c
```

• Output:

```
/usr/softtune/lib/896/include/stdio.h
/usr/softtune/lib/896/include/stddef.h
/usr/softtune/lib/896/include/stdarg.h
./head.h
```

● -I dir, -XI

The -I option changes the manner of header file search so that the directory specified by dir will be searched prior to the standard directory. The standard directory is \${INC896}.

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 angle 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.

#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]

```
>fcc896s -U m -D n -D m file.c -cpu MB89P935B
```

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

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:

```
fcc896s -cmsg -S -cpu MB89P935B 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]

```
>fcc896s -help
```

Various command option lists are output 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.

[Output Example]

- Input:

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

```
fcc896s -INF lineno -S -cpu MB89P935B sample.c
```
- Output:

```
_func:
;;;      sample.c, line 1
L_func:
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]

```
>fcc896s -INF list -c -cpu MB89P935B sample.c
```

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

```
fcc896s -INF srcin -S -cpu MB89P935B sample.c
```
- Output:

```
func:
;;;      void func(void){}
L_func:
RET
```

● -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 MUSC 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:

```
extern void sub(void);  
void func(void){sub();}
```

• Operation:

```
fcc896s -INF stack -S -cpu MB89P935B sample.c
```

• Output:

```
@sample.c  
#   E=Extern   S=Static   I=Interrupt  
#   {Stack}    {E|S|I}   {function name}[A]  
#           ->   {E|S}    {call function}  
#           ...  
#  
#           2      E      _func  
#           ->   E      _sub
```

● -o path name, -Xo

The -o path name option uses the path name 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]

```
>fcc896s -o output.asm -S -cpu MB89P935B sample.c
```

The sample.c preprocessing and compiling process result are output 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 the warning level and item to be warned, refer to Table 3.5-3.

[Output Example]

- Input:

```
const int a;
```
- Operation:

```
fcc896s -w 5 -S -cpu MB89P935B sample.c
```
- Output:

```
*** a.c(1) W1219C: 'const' a is not initialized.
```

Table 3.5-3 Warning Item at Each Warning Level (1 / 2)

Warning level	Warning item
Level 0	Warning-type diagnostic message is not generated.
Level 1	A basic warning-type diagnostic messages are generated.
Level 2	In addition to level 1, the following warning-type diagnostic messages 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	In addition to level 2, the following warning-type diagnostic messages are generated. When there is no return in the function which should return the value, a warning is generated. When the value is not specified for return by the function which should return the value, a warning is generated. Warning of pragma which cannot be recognized is generated. When the variable and the constant are compared in the comparison operation, a warning of the range of the value of the constant is generated.
Level 4	In addition to level 3, the following warning-type diagnostic messages are generated. When the extern function is declared in the block, a warning is generated. When the struct/union is not defined in the external declaration of the struct/union array, a warning is generated. When not the relational expression but the assignment expression, etc. are described in the place where the conditional expression is expected, a warning is generated. When the address of the auto variable is used as a return value of the function, a warning is generated.

Table 3.5-3 Warning Item at Each Warning Level (2 / 2)

Warning level	Warning item
Level 5	In addition to level 4, the following warning-type diagnostic messages are generated. When there is an implicit int type declaration, a warning is generated. When there is no prototype declaration of the function, a warning is generated. When the constant is described in the condition expression, a warning is generated. When there is an implicit int type declaration of the parameter, a warning is generated. When the declaration overload the declaration before, a warning is generated. When the comma continues at enum member's end, a warning is generated. When there is no initial value in the declaration with const, a warning is generated. When the address of the variable is compared with 0, a warning is generated. When the type is defined in the cast expression, a warning is generated. When register is specified for struct, union, and the array variable declaration, a warning is generated.
Level 6	In addition to level 5, the following warning-type diagnostic messages are generated. When there is any switch statement which does not have a default label, a warning is generated. When promotion to the unsigned int type exists, a warning is generated. When promotion to the int type exists, a warning is generated. When the pointer type is necessary for the argument, a warning is generated. When the type of the return value of the function is not expanded, a warning is generated. When the escape sequence '\x' is specified, a warning is generated. When the size of the type of the parameter of a prototype declaration is different from that of the argument, a warning is generated. When the sign of the type of the parameter of a prototype declaration is different from that of the argument, a warning is generated. When there is data for which an initial value is not specified, a warning is generated.
Level 7	In addition to level 6, the following warning-type diagnostic messages are generated. When the int type is used, a warning is generated. When the bit field is neither int, signed int nor unsigned int type, a warning is generated. When the kind of the type of the parameter of a prototype declaration is different from that of the argument, a warning is generated.
Level 8	In addition to level 7, the following warning-type diagnostic message is generated. When the function is called with a pointer to the function, a warning is generated.

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 | n}

The level of the language specification that the compiler and the pre-processor use is set.

Table 3.5-4 shows the level of the language specification when the option is specified.

Table 3.5-4 Level of Language Specification Interpreted when Option is Specified

option	Level of language specification
-Jc	Using the strict ANSI standard
-Ja	Using the Extended Language Specifications
-Jn	Using the Extended Language Specifications and ANSI standard non-conforming specifications

When -Jc is specified, warning messages are issued to the use of the extended language specifications.

Please refer to "CHAPTER 5 EXTENDED LANGUAGE SPECIFICATIONS" for the extended language specifications for -Ja.

Please refer to "APPENDIX C ANSI Standard Non-Conforming Specifications" for the ANSI standard non-conforming specifications for -Jn.

If the -J {a | c | n} option is not specified, -Ja applies.

[Example]

```
>fcc896s -J a file1.c -J c file2.c -cpu MB89P935B
```

The -Jc option becomes valid so that files named file1.c and file2.c are interpreted in strict compliance with the ANSI specifications.

Note:

When object compiled with -Jn option and object compiled without -Jn option are mixed, the operation is not guaranteed.

● -K NO_REDUCED_TYPE_OPERATION, -K REDUCED_TYPE_OPERATION

The NO_REDUCED_TYPE_OPERATION suboption invalidates the following ANSI standard non-conforming specifications.

- The integral promotions are not applied.
- The type of the integer constant has a minimum type.
- The type of the enumeration type has a minimum type.
- The type of the truth-value of the operation result has type char.

This option is effective only when specified with -Jn option.

When the -Jn option is not specified, a warning message is output.

The REDUCED_TYPE_OPERATION suboption cancels the NO_REDUCED_TYPE_OPERATION suboption.

Note:

When object compiled with -K NO_REDUCED_TYPE_OPERATION option and object compiled without -K NO_REDUCED_TYPE_OPERATION option are mixed, the operation is not guaranteed.

● -K NO_REDUCED_TYPE_ARGUMENT, -K REDUCED_TYPE_ARGUMENT

The NO_REDUCED_TYPE_ARGUMENT suboption invalidates the following ANSI standard non-conforming specification.

- The default argument promotions are not applied.

This option is effective only when specified with -Jn option.

When the -Jn option is not specified, a warning message is output.

The REDUCED_TYPE_ARGUMENT suboption cancels the NO_REDUCED_TYPE_ARGUMENT suboption.

Note:

When object compiled with -K NO_REDUCED_TYPE_ARGUMENT option and object compiled without -K NO_REDUCED_TYPE_ARGUMENT option are mixed, the operation is not guaranteed.

● -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:


```
fcc896s -K fconst -cpu MB89P935B -S sample.c
```
- Output:


```
_func:
        MOVW    A, _f2+2
        PUSHW   A
        MOVW    A, _f2
        PUSHW   A
        MOVW    A, #0
        PUSHW   A
        MOVW    A, #16256
        PUSHW   A
        CALL    LFADD
        PUPW    A
        MOVW    _f1, A
        POPW    A
        MOVW    _f1+2, A

L_func:
        RET
```

● **-K NOINTLIB, -K INTLIB**

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:


```
fcc896s -K nointlib -cpu MB89P935B -S sample.c
```
- Output:


```
_func:
        CALL    __DI

L_func:
        RET
```

● **-K NOVOLATILE, -K VOLATILE**

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]

```
>fcc896s -K novolatile -S -O -cpu MB89P935B 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.

● **-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:

```
fcc896s -K schar -cpu MB89P935B -S sample.c
```

• Output:

```
MOVW    A, #0
MOV     A, _c; Code-extended
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 `SOFTUNE REALOS/896 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 "sctype.h"
void func(void){ ext_tsk();}
```
- Operation:


```
fcc896s -K realos -cpu MB89P935B -S sample.c
```
- Output:


```
CALL    ext_tsk
```

● -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:


```
fcc896s -K sbit -cpu MB89P935B -S sample.c
```
- Output:


```
MOV      A, _st+1
MOVW     A, #15
CALL     LSHLW
MOVW     A, #15
CALL     LSHRW           :Code-extended
MOVW     _data, A
```

● -kanji {SJIS | EUC} option

If Japanese is entered in a program, the code system for the used Japanese is specified.

Japanese including half-size kana can be entered in program comments and character strings. The compiler identifies the code system for Japanese description based on this option. SJIS means that the Shift JIS code system is used, and EUC means that the EUC code system is used. When this option is omitted, -kanji SJIS is used for HP-UX and Windows, and -kanji EUC is used for Solaris.

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 -K SIZE option 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 -K SPEED option 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, the shorter the generated object execution time but the longer the compilation time. Note that higher optimization level contains lower optimization level functions.

One of the following levels is to be specified. When no level is specified, -O2 applies.

-0: Optimization Level 0

Minimum optimization will be effected as not obstructing the C source level debugging. This level is equivalent to cases where the -O 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.

- **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.

[Output Example]

- Input:

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

- Operation:

```
fcc896s -K addsp -cpu MB89P935B -S sample.c
```

- Output:

```
MOVW    A, _i
PUSHW   A
CALL    _sub
MOVW    A, _i
PUSHW   A
CALL    _sub
POPW    A           ; Releasing argument areas synthesized
POPW    A
```

● -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. When optimization level is greater than 1 or equal, this option takes effect.

The NOEOPT suboption cancels the EOPT suboption.

[Output Example]

- Input:

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

- Operation:

```
fcc896s -K eopt -O -cpu MB89P935B -S sample.c
```

- Output:

```
MOVW    A, @IX+4
MOVW    A, @IX+6
CLRC
ADDCW   A           ; Order of arithmetic operation replaced
MOVW    _i, A
```

● -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. When optimization level is greater than 1 or equal, this option takes effect.

The NOLIB suboption cancels the LIB suboption.

[Output Example]

- Input:


```
extern int i;
void func(void){
    i=strlen("ABC");
}
```
- Operation:


```
fcc896s -K lib -O -cpu MB89P935B -S sample.c
```
- Output:


```
MOVW    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 of different pointers.

When optimization level is greater than 1 or equal, this option takes effect.

The language specification permits different pointers to point the same area. 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:


```
fcc896s -K noalias -O -cpu MB89P935B -S sample.c
```
- Output:


```
MOVW    A, _i
INCW    A
MOVW    @IX-2, A
MOVW    A, @IX+4
MOVW    @A, T
MOVW    A, @IX-2
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, setjmp function calling, and other conditions. When optimization level is greater than 1 or equal, this option takes effect.

The -Xx option cancels the -x option.

[Output Example]

• Input:

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

• Operation:

```
fcc896s -cpu MB89P935B -O -x sub -S sample.c
```

• Output:

```
_func:
        MOVN    A, #1
        MOVW    _a, A
L_func:
        RET
```

● -xauto [size], -Xxauto

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, setjmp function calling, and other conditions.

If the size entry is omitted, the value 30 is assumed to be specified. When optimization level is greater than 1 or equal, this option takes effect.

The -Xxauto option cancels the -xauto option.

● -K ARRAY, -K NOARRAY

The -K ARRAY optimizes the array element reference code(e.g. a[i]++;). When optimization level is greater than 1 or equal, this option takes effect. 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.

3.5.6 Output Object Related Options

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]

```
>fcc896s -S -cpu MB89P935B sample.c
```

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

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

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

For compiler output section names, see “4.1 Section Structure of fcc896s Command”. 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.

Note:

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:

```
fcc896s -s CODE=PROGRAM, CODE, 0x1000 -S -cpu MB89P935B sample.c
```

- Output:

```
.SECTION          PROGRAM, CODE, LOCATE=H'1000
.GLOBAL __func
__func:
L_func:
RET
```


3.5.7 Debug Information Related Options

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

■ Debug Information Related Options

- -g, -Xg

-g option adds debug information to the object file.

The -Xg option cancels -g option.

When optimization options are specified, please note the following issues.

- The breakpoint might not be able to be set.
Because the line might be moved or deleted, the breakpoint might not be able to be set.
- Plural breakpoints might be set at once.
Because the lines from which the instructions are deleted might be consecutive, plural breakpoints might be set at once.
- It might not stop at the breakpoint.
Because the instructions of the line might be moved or deleted, it might not stop at the breakpoint.
- The value of the variable displayed in the watch window might not be correct.
Because the instruction that stores the value of the variable might be moved, the timing to be updated might not correspond to the order of the C source.
Because one register might be assigned to two or more variables, the timing to be updated might not correspond to the order of the C source.
- The local variable and the parameter might not be able to be referred to by the debugger.
The optimized local variable and parameter might not be able to be referred to by the debugger.
- The CALL STACK function and the STEP OUT function might not be able to be used by the debugger.
When the prologue/epilogue code of the function is optimized, the CALL STACK function (SHOW CALLS command) and the STEP OUT function (GO /RETURN command) cannot be used by the debugger.
- When the function inlining (-x func or -xauto option) is specified, the C source displayed in the source window might not be correct.

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 path name to dir

c: Changes the compiler path name to dir

a: Changes the assembler path name to dir

l: Changes the linker path name to dir

[Example]

```
>fcc896s file.c -Yp,\home\newlib -cpu MB89P935B
```

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
- l: Linker

[Example]

```
>fcc896s -Ta,-lf,asmlist file.c -cpu MB89P935B
```

Sequentially passes arguments -l and asmlist to the assembler. Therefore, the assemble list asmlist will be generated as a result of command execution.

3.5.9 Linkage Related Options

This section describes the options related to linkage.

■ Linkage Related Options

- **-e name, -Xe**

The -e option sets the entry symbol to name at linking. The -Xe option cancels the -e option. Since the option definition is usually provided in the startup routine, this option does not have to be specified.

For details of the option, refer to the Linkage Kit Manual.

- **-L path1 [, path2 ...], -XL**

The -L option adds path to the library path used at linking to search for a library to be linked.

If the option is not specified, \${LIB896} is selected automatically.

The -XL option cancels the -L option.

For details of the option, refer to the Linkage Kit Manual.

- **-l lib1 [, lib2 ...], -Xl**

The -l option specifies the name (lib) of the library to be linked at linking. If the extension entry is omitted, the .lib extension is added automatically.

The -Xl option cancels the -l option.

For the objects output by the compiler, by default, "lib896.lib" are set as the names of the libraries to be linked.

For the details of the option, refer to the Linkage Kit Manual.

- **-m, -Xm**

The -m option generates a map file at linking.

A map file output with a file name to the .map extension is generated in the current directory.

The -Xm option cancels the -m option.

- **-ra name=start/end, -Xra**

The -ra option specifies the RAM area at linking.

The -Xra option cancels the -ra option.

For details of the option, refer to the Linkage Kit Manual.

- **-ro name = start/end, -Xro**

The -ro option specifies the ROM area at linking.

The -Xro option cancels the -ro option.

For details of the option, refer to the Linkage Kit Manual.



- -sc param, -Xsc

The -sc option specifies the section arrangement at linking. The -Xsc option cancels the -sc option. If the option is not specified, -sc IOPORT=0,*=0x1000 is selected automatically.

For details of the option, refer to the Linkage Kit Manual.

- -startup filename, -Xstartup

The -startup option selects filename as the object file name of the startup routine to be linked at linking.

If the option is not specified, "%FETOOL%\lib\896\startup.obj" is selected automatically.

The -Xstartup option cancels the -startup option.

For details of the startup routine, see "CHAPTER 6 EXECUTION ENVIRONMENT".

3.5.10 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”.

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]

```
-I /usr/include # Include specifying
-D F2MC8L      # 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.

OS	Kanji character code which can be used
Windows	ShiftJIS
Solaris2.x	EUC
HP-UX10.x	ShiftJIS

■ 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]

```
-I /usr/include    # Include specifying
-D F2MC8L          /* 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]

`${OPT896}/fcc896.opt`

[For Windows]

`%OPT896%\fcc896.opt`

The default option file name is "fcc896.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

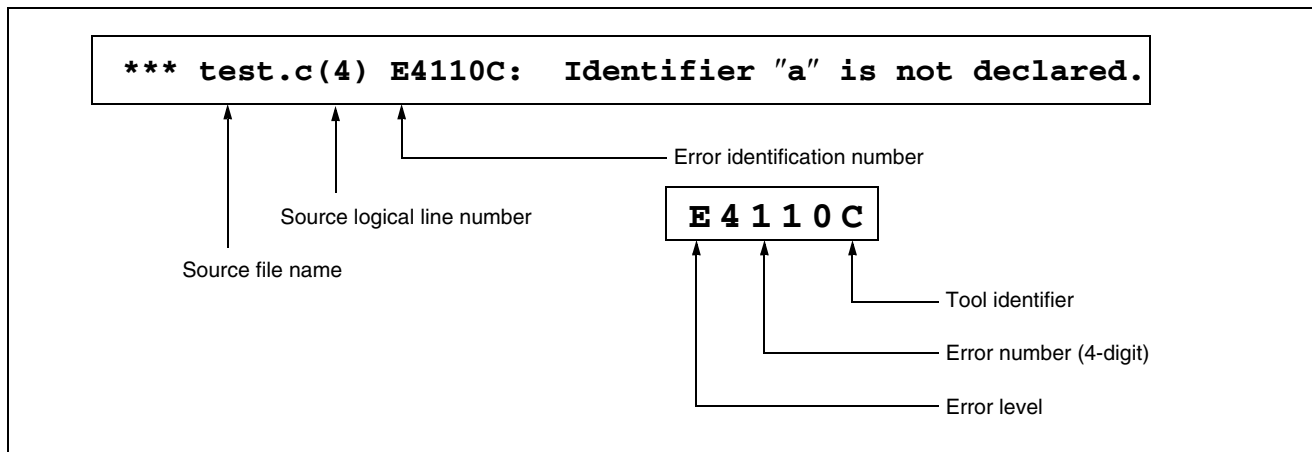
When an error is found in a source program or a condition which does not constitute a substantial error but requires attention is encountered, diagnostic messages may be generated at the time of translation.

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.

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
- L: Linker

■ Error Level

The error level represents the diagnostic check result type.

Table 3.7-1 shows the relationship between various error levels and return codes and their meanings.

Table 3.7-1 Relationship between Error Levels and Return Codes

Error Level	Return Code	Meaning
I	0	Indicates a condition which does not constitute an error but requires attention.
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 of fcc896s Command
- 4.2 Generation Rules for Names Used by Compiler
- 4.3 Boundary Alignment of fcc896s Command
- 4.4 Bit Field of fcc896s Command
- 4.5 Structure/Union of fcc896s Command
- 4.6 Function Call Interface of fcc896s Command
- 4.7 Interrupt Function Call Interface of fcc896s Command



4.1 Section Structure of fcc896s Command

The fcc896s command has the following eight sections:

- Code section
- Initialized section
- Constant section
- Data section
- Initialized direct section
- Direct section
- I/O section
- Vector section

■ fcc896s Command Section Structure

Table 4.1-1 shows the sections to be generated by the compiler and their meanings.

Table 4.1-1 fcc896s Command Section List

No.	Section Type	Section Name	Type	Boundary Alignment [Byte]	Write	Initial Value
1	Code section	CODE	CODE	1	Disabled	Provided
2	Initialized section	INIT	DATA	1	Enabled	Provided
3	Constant section	CONST	CONST	1	Disabled	Provided
4	Data section	DATA	DATA	1	Enabled	Not provided
5	Initialized direct section	DIRINIT	DIR	1	Enabled	Provided
6	Direct section	DIRDATA	DIR	1	Enabled	Not provided
7	I/O section	IO	IO	1	Enabled	Not provided
8	Vector section	INTVECT	CONST	1	Disabled	Provided

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.

(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.

(3) Constant section

Stores the write-protected initial value attached variable area. For the C language, this section corresponds to the area for const attribute attached external variables, static external variables, and static internal variables.

(4) 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 attribute), static external variables, and static internal variables.

(5) 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 attribute.

The default section name is DIRINIT.

(6) 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 attribute), static external variables, and static internal variables.

The default section name is DIRDATA.

(7) 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 attribute), static external variables, and static internal variables.

The default section name is IO.

(8) Vector section

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

The default section name is INTVECT.

4.2 Generation Rules for Names Used by Compiler

This section describes the rules for the names used by the compiler.

■ Generation Rules for Names Used by Compiler

Table 4.2-1 shows the relationship between the names generated by the compiler and the C language.

Table 4.2-1 Label Generation Rules

C Language Counterpart	Label Generated by Compiler
Function name	_function name
External variable name	_external variable name
Static variable name	LI_no
Local variable name	—
Virtual argument name	—
Character string, derived type	LS_no
Automatic variable initial value	LS_no
Target location label	L_no

Remark: The compiler internal generation number is placed at the no position.

4.3 Boundary Alignment of fcc896s Command

This section describes the standard data type and boundary alignment. Table 4.3-1 shows the assignment rules.

■ fcc896s Command Boundary Alignment

Table 4.3-1 fcc896s Command Variable Assignment Rules

Variable Type	Assignment Size [Byte]	Boundary Alignment [Byte]
char	1	1
signed char	1	1
unsigned char	1	1
short	2	1
unsigned short	2	1
int	2	1
unsigned int	2	1
long	4	1
unsigned long	4	1
float	4	1
double	8	1
long double	8	1
Pointer/address	2	1
Structure/union	Explained later	Explained later



4.4 Bit Field of fcc896s Command

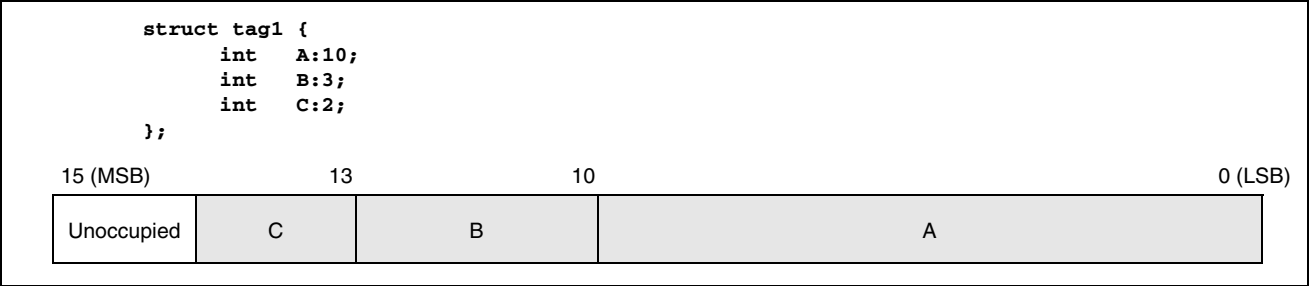
This section describes the bit field data size and boundary alignment for the `fcc896s` command.

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.

■ fcc896s Command Bit Field

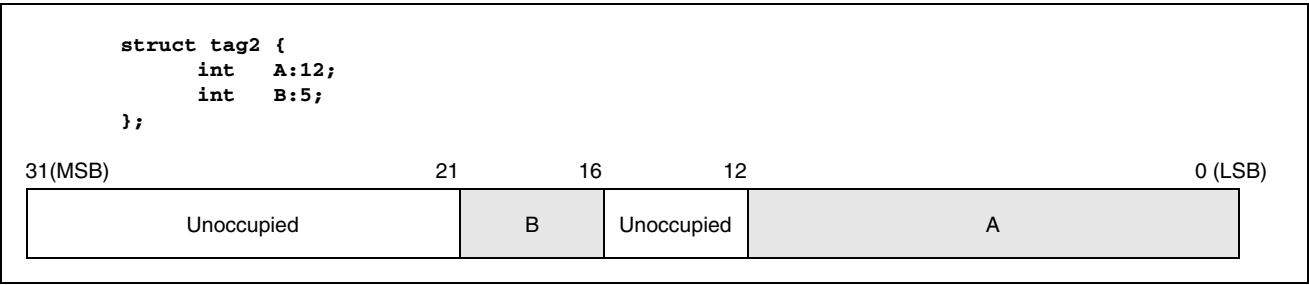
Consecutive bit field data are packed at consecutive bits having the same storage unit, beginning with the LSB and continuing toward the MSB. An example is shown in Figure 4.4-1.

Figure 4.4-1 Example 1 of Bit Field Data Size and Boundary Alignment for fcc896s Command



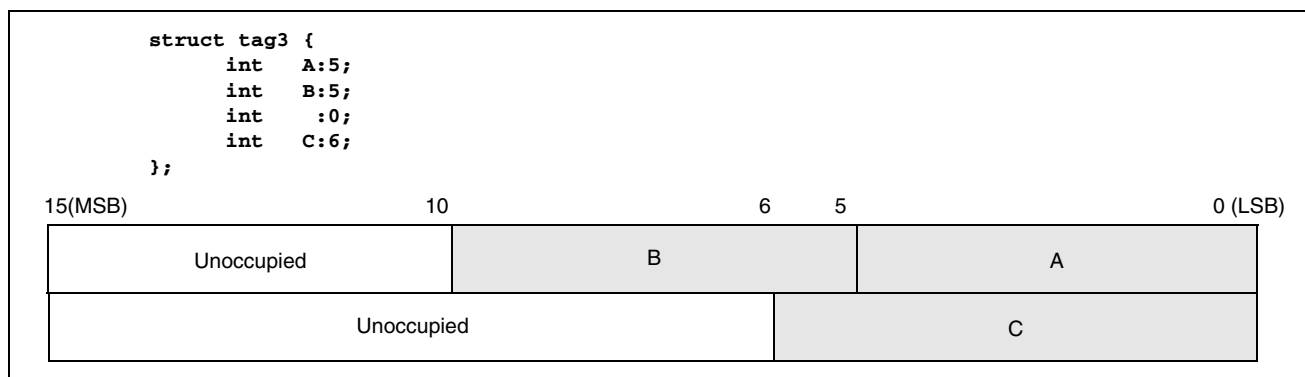
If a field to be assigned lies over a bit field type boundary, its assignment is completed by aligning it with a boundary suitable for the type. An example is shown in Figure 4.4-2.

Figure 4.4-2 Example 2 of Bit Field Data Size and Boundary Alignment for fcc896s Command



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 4.4-3.

Figure 4.4-3 Example 3 of Bit Field Data Size and Boundary Alignment for fcc896s Command



4.5 Structure/Union of fcc896s Command

This section describes the structure/union data size and boundary alignment for the `fcc896s` command. The structure data size is equal to total of the member size. The union data size is equal to the size of the maximum member.

■ fcc896s Command Structure/Union

```
struct st1 { char A; }      →sizeof(st1) = 1 BYTE
struct st2 { short A; }    →sizeof(st2) = 2 BYTES
struct st3 { char A; short B; } →sizeof(st3) = 3 BYTES
struct st4 { int A; char B; } →sizeof(st4) = 3 BYTES
```

4.6 Function Call Interface of fcc896s Command

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.

■ fcc896s Command Function Call Interface

- 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 or 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.

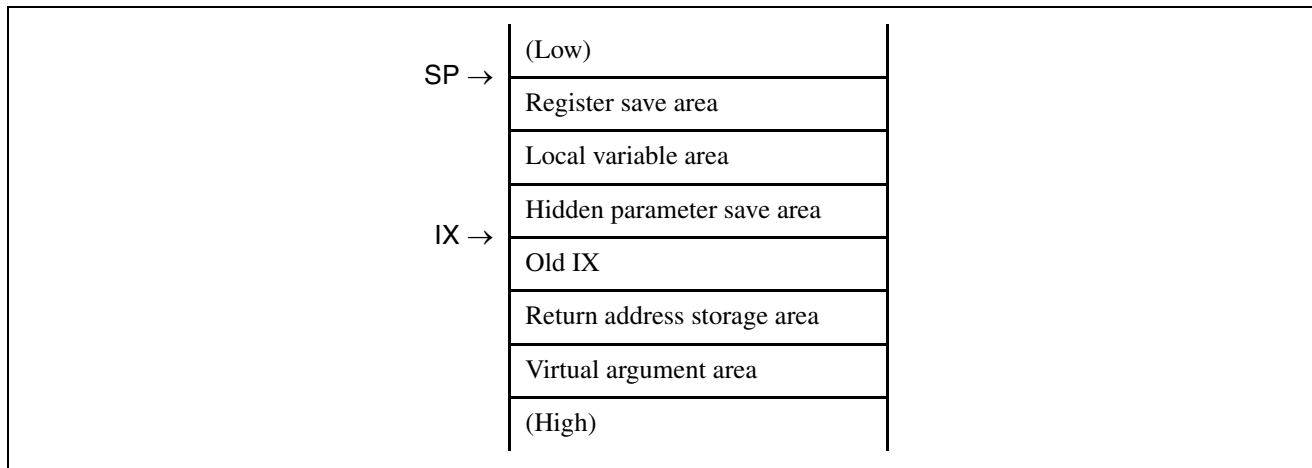
4.6.1 Stack Frame of fcc896s Command

The standard linkage regulations prescribe the stack frame construction.

■ fcc896s Command Stack Frame

The stack pointer (SP) always indicates the lowest order of the stack frame. Its address value always represents the word boundary. Figure 4.6-1 shows the standard function stack frame status.

Figure 4.6-1 fcc896s Command Stack Frame



(1) 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.

(2) Local variable area

This is the area for local variables and temporary variables.

(3) Hidden parameter save area

This area stores the start address of the return value storage area for a structure/union return function.

When a structure/union is used as the return value, the caller function stores the return value storage area start address in register EP and passes it to the callee function.

The callee function interprets the address stored in the R4 as the return value storage area start address.

When register EP needs to be saved into memory, the callee function saves it in the hidden parameter save area. This area is not secured when the save operation is not needed.

(4) Old IX

This area stores the frame pointer (IX) 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/virtual argument 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 virtual argument area.

For details, see “4.6.2 Argument of fcc896s Command”.

4.6.2 Argument of fcc896s Command

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.

■ fcc896s Command Argument

Figure 4.6-2 shows an example of argument transfer relative to the callee function.

Figure 4.6-2 Argument Format for Standard Linkage Regulation of fcc896s Command

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

(Low)
1
Unoccupied
st
Unoccupied
2
(High)

4.6.3 Argument Extension Format of fcc896s Command

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.

■ fcc896s Command Argument Extension Format

Table 4.6-1 shows the argument extension format.

Table 4.6-1 fcc896s Command Argument Extension Format

Argument Type	Extended Type* ¹	Stack Storage Size [Byte]
char	int	2
signed char	int	2
unsigned char	int	2
short	No extension	2
unsigned short	No extension	2
int	No extension	2
unsigned int	No extension	2
long	No extension	4
unsigned long	No extension	4
float	double	8
double	No extension	8
long double	No extension	8
Pointer/address	No extension	2
Structure/union	*2	*2

*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.

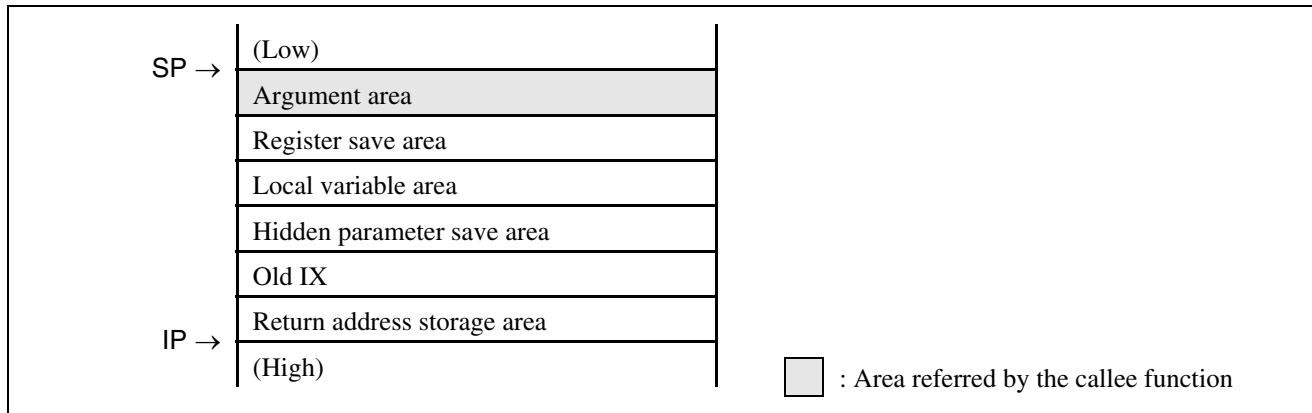
4.6.4 Calling Procedure of fcc896s Command

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

■ fcc896s Command Calling Procedure

Figure 4.6-3 shows the stack frame prevailing at calling in compliance with the standard linkage regulations.

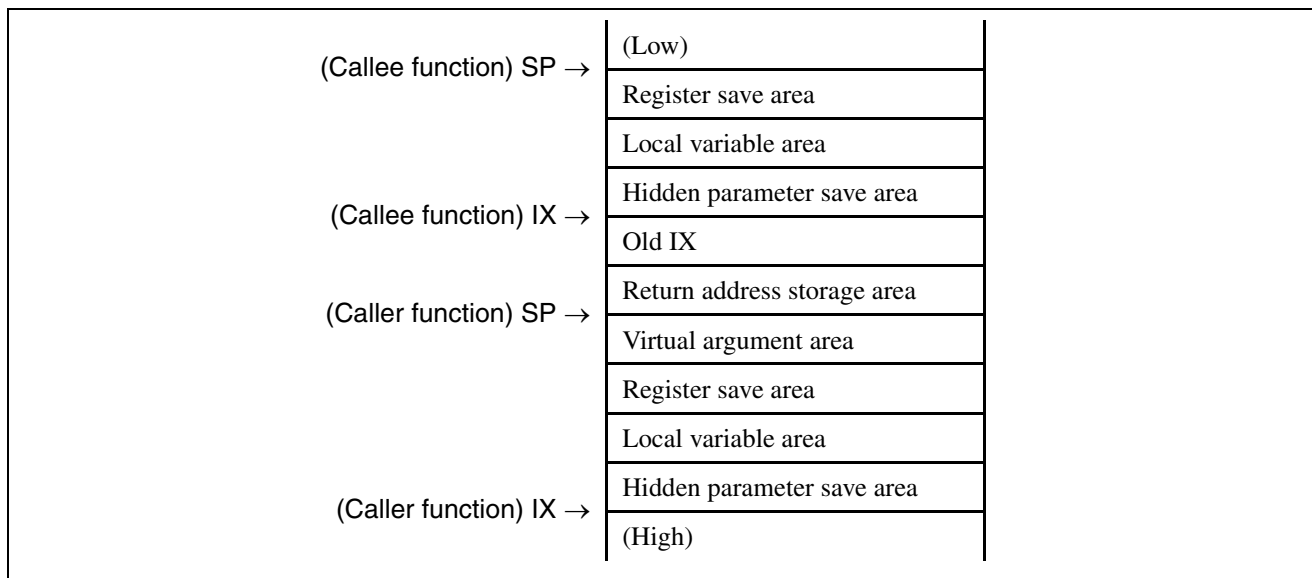
Figure 4.6-3 Stack Frame Prevailing at Calling in Compliance with fcc896s Command Standard Linkage Regulations



The callee function saves the caller function frame pointer (IX) 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 4.6-4 shows the stack frame that is created by the callee function in compliance with the standard linkage regulations.

Figure 4.6-4 Stack Frame Created by Callee Function in Compliance with fcc896s Command Standard Linkage Regulations



4.6.5 Register of fcc896s Command

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

■ fcc896s Command Register Guarantee

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

- General-purpose registers R2 to R7, IX and SP

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.

■ fcc896s Command Register Setup

Table 4.6-2 shows the register regulations for function call and return periods.

Table 4.6-2 Register Regulations for fcc896s Command Function Call and Return Periods

Register	Call Period	Return Period
EP	Return value area address	Return value*
A and T	Not stipulated	Not stipulated
R0 and R1	Not stipulated	Not stipulated
R2 to R7	Not stipulated	Call period value guaranteed
IX	Frame pointer	Call period value guaranteed
SP	Stack pointer	Call period value guaranteed

*: There are no stipulations where a function without the return value is called or a function having a structure/union/long/double/long double type return value is called.

4.6.6 Return Value of fcc896s Command

Table 4.6-3 shows the return value interface stated in the standard linkage regulations.

■ fcc896s Command Return Value

Table 4.6-3 fcc896s Command Return Value Interface Stated in Standard Linkage Regulations

Return Value Type	Return Value Interface
void	None
char	EP
signed char	EP
unsigned char	EP
short	EP
unsigned short	EP
int	EP
unsigned int	EP
long	EP*
unsigned long	EP*
float	EP*
double	EP*
long double	EP*
Pointer/address	EP
Structure/union	EP*

*: The caller function stores the start address of the return value storage area into EP and then passes it to the callee function. The callee function interprets EP 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.

4.7 Interrupt Function Call Interface of fcc896s Command

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.

■ fcc896s Command Interrupt Function Call Interface

- 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.

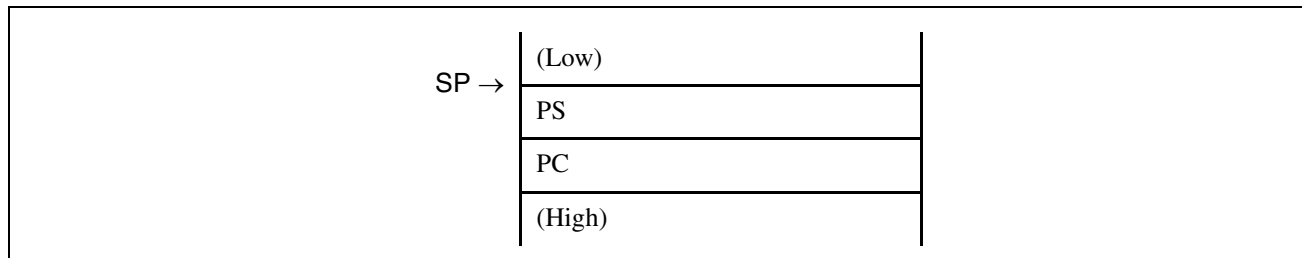
4.7.1 Interrupt Stack Frame of fcc896s Command

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

■ Interrupt Stack Frame

Figure 4.7-1 shows the interrupt stack frame status prevailing immediately after interrupt generation.

Figure 4.7-1 fcc896s Command Interrupt Stack Frame



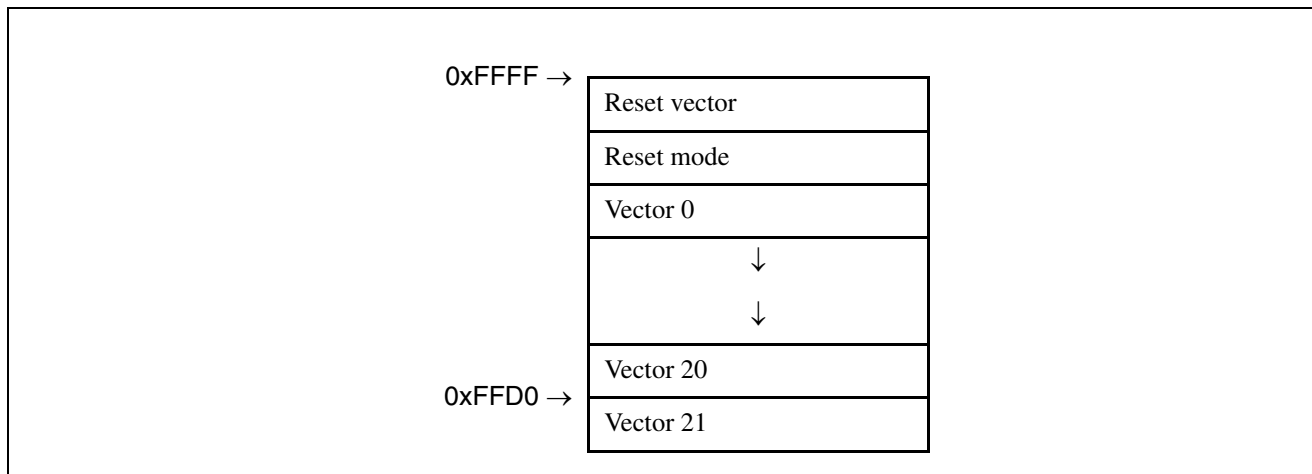
4.7.2 Interrupt Function Calling Procedure of fcc896s Command

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.

■ Interrupt Function Calling Procedure

Figure 4.7-2 shows an example of interrupt vector table.

Figure 4.7-2 fcc896s Command 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 In-line Expansion Specifying Function
- 5.6 Section Name Change Function
- 5.7 Register Bank Number Setup Function
- 5.8 Interrupt Level Setup Function
- 5.9 No-Register-Save Interrupt Func. Function
- 5.10 Built-in Function
- 5.11 Predefined Macros
- 5.12 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 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, a part of optimization is stopped even when the -O optimization option is specified.

[Output Example]

- Input:

```
/* When written inside the function */
extern int temp;
sample() {
    __asm("    MOVW    A, #1");
    __asm("    MOVW    _temp, A");
}
/* When written outside the function */
__asm("        .SECTION        DATA, DATA, ALIGN=1");
__asm("        .GLOBAL _a");
__asm("    _a:    .RES.H    1");
```

- Output:

```
.SECTION          CODE, CODE, ALIGN=1
.GLOBAL _sample
_sample:
    MOVW    A, #1
    MOVW    _temp, A
L_sample:
    RET
.SECTION          DATA, DATA, ALIGN=1
.GLOBAL _a
_a:      .RES.H 1
```

■ Description by Pragma Instruction

The description between `#pragma asm` and `#pragma endasm` directly is expanded as the assembler. 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 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, a part of optimization is stopped even when the `-O` optimization option is specified.

[Output Example]

- Input:

```
/* When written inside the function */
extern int temp;
sample(){
#pragma asm
    MOVW    A, #1
    MOVW    _temp, A
#pragma endasm
}
/* When written outside the function */
#pragma asm
    .SECTION          DATA, DATA, ALIGN=1
    .GLOBAL _a
_a:      .RES.H 1
#pragma endasm
```



- Output:

```
.SECTION          CODE, CODE, ALIGN=1
.GLOBAL _sample
_sample:
    MOVN    A, #1
    MOVW    _temp, A
L_sample:
    RET
.SECTION          DATA, DATA, ALIGN=1
.GLOBAL _a
_a:      .RES.H  1
```


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]

```
void __DI(void);
```

[Explanation]

Expands the interrupt masking code

[Output Example]

- Input:

```
__DI ( ) ;
```

- Output:

```
CLRI
```

■ Interrupt Mask Disable Function

[General Format]

```
void __EI(void);
```

[Explanation]

Expands the interrupt masking disable code

[Output Example]

- Input:

```
__EI ( ) ;
```

- Output:

```
SETI
```

■ Interrupt Level Setup Function

[General Format]

```
void __set_il(int level);
```

[Explanation]

Expands the code for changing the interrupt level to level

[Output Example]

- Input:

```
__set_il (2) ;
```

- Output:

```
MOVW    A, PS
```

```
CLRI
```

```
MOVW    A, #207
```



```
ANDW    A
MOVW    A, #32
ORW     A
MOVW    PS, A
```

■ 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:
        PUSHW    A
        XCHW     A, T
        PUSHW    A
        MOVW     A, EP
        PUSHW    A
        MOV      A, R0
        SWAP
        MOV      A, R1
        PUSHW    A
L_sample:
        POPW     A
        MOV      R1, A
        SWAP
        MOV      R0, A
        POPW     A
        MOVW     EP, A
        POPW     A
        XCHW     A, T
        POPW     A
        RETI
```

■ Interrupt Vector Table Generation Function

[General Format]

`#pragma intvect Interrupt function name Vector number`

`#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 output.

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.

No reset vector and reset mode are included in the vector table. They must be defined separately by the `asm` statement.



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 to operate an I/O area defined at addresses between 0x00 and 0xff can be defined by specifying the `__io` type qualifier.

Since a highly-efficient 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 the area above 0xff, use the volatile type qualifier.

The initial value cannot be specified for variables with the `__io` type qualifier.

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

When the `__io` type qualifier is specified for an automatic variable, the variable is not treated as a variable positioned in the I/O area but an automatic variable for which the volatile 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'10
.GLOBAL _a
_a:
.RES.H 1
.SECTION          CODE, CODE, ALIGN=1
.GLOBAL _func
_func:
    MOVW    A, #1
    MOVW    _a, A
L_func:
    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 there is 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 `fcc896s` command, to make accessible the section (`DIRDATA/DIRINIT`) generated by `__direct` type qualifying the variable, the sections must be arranged in the `0x00` to `0xFF` range. The area in this range is also used as the I/O area, so the sections should be arranged in an area that is not used as the I/O area.

[Output Example]

- Input:

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

- Output:

```
.SECTION          DIRDATA, DIR, ALIGN=1
.GLOBAL _p
_p:
.RES.H 1
.SECTION          CODE, CODE, ALIGN=1
.GLOBAL _sample
_sample:
    MOVW    A, #1
    MOVW    _p, A
L_sample:
    RET
```

5.5 In-line Expansion Specifying Function

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]

`#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, 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.

5.6 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 =.)

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

For the section name output by the compiler, see “4.1 Section Structure of fcc896s Command”. 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.

Note:

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'FF
.GLOBAL _main
_main:
L_main:
RET
```

[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 section cannot specify the #pragma segment for the fcc896s command.

[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          ram1, DATA, ALIGN=1
.GLOBAL _a1
_a1:
    .RES.H 1
    .SECTION          ram2, DATA, ALIGN=1
.GLOBAL _a2
_a2:
    .RES.H 1
    .SECTION          program1, CODE, ALIGN=1
.GLOBAL _func1
_func1:
    RET
    .SECTION          program2, CODE, ALIGN=1
.GLOBAL _func2
_func2:
    RET
```


Note:

#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

Variable/function names	Section name
var	const1
func	program2



5.7 Register Bank Number Setup Function

This function is used to specify the register bank that the function uses.

■ Register Bank Number Setup Function

[General Format]

```
#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:

```
#pragma register(2)
void func(void){}
#pragma noregister
```

- Output:

```
_func:
        MOVW    A, PS
        SWAP
        MOV     A, #16
        SWAP
        MOVW    PS, A
L_func:
        RET
```

5.8 Interrupt Level Setup Function

This function is used to set the function interrupt level.

■ Interrupt Level Setup Function

[General Format]

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

[Explanation]

#pragma ilm specifies the interrupt level for the subsequently defined function.

#pragma noilm clears the interrupt level specifying. An integer constant between 0 and 3 can be specified in the NUM position.

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:

```
#pragma ilm(1)
void func(void){}
#pragma noilm
_func:
        MOV     ILM, #1
        LINK    #0
        UNLINK
        RET
```

- Output:

```
_func:
        MOVW    A, PS
        AND     A, #207
        OR      A, #16
        MOVW    PS, A
L_func:
        RET
```



5.9 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:
PUSHW    A
XCHW     A, T
PUSHW    A
MOVW     A, EP
PUSHW    A
MOVW     A, PS
SWAP
MOV      A, #40
SWAP
MOVW     PS, A
CALL     _sub
L_func:
POPW     A
MOVW     EP, A
POPW     A
XCHW     A, T
POPW     A
RETI
```

5.10 Built-in Function

The following built-in function is available.

- `__wait_nop`
-

■ `__wait_nop` Built-in Function

[General Format]

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

[Explanation]

To properly time 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:

```
void sample(void){__wait_nop();}
```
- Output:

```
_sample:
        NOP
L_sample:
        RET
```

5.11 Predefined Macros

This section describes the macro names predefined by the compiler.

■ Macros Stipulated by ANSI Standard

The ANSI standard stipulates the following macros.

Macro Name	Description
__LINE__	Defines line number of current source line
__FILE__	Defines source file name
__DATA__	Defines source file translation date
__TIME__	Defines source file translation time
__STDC__	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.

■ Macros Predefined by fcc896s Command

The fcc896s command predefines the following macros.

Macro Name	Description
__COMPILER_FCC896__	Selects 1 as definition
__CPU_MB number__	Selects MB number specified by the -cpu option as definition
__CPU_8L__ __CPU_8FX__	Selects 1 as definition for macro of certain series name in accordance with MB number specified by the -cpu option

5.12 Limitations on Compiler Translation

Table 5.12-1 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 5.12-1 List of Translation Limitations (1 / 2)

No.	Function	ANSI Standard	Compiler
1	Count of nesting levels for a compound statement, repetition control structure, and selection control structure	15	∞
2	Count of nesting levels for condition incorporation	8	∞
3	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	12	∞
4	Count of nests provided by parentheses for one complete declarator	31	∞
5	Count of nest expressions provided by parentheses for one complete expression	32	∞
6	Count of valid leading characters of internal identifier or macro name	31	∞
7	Count of valid leading characters of external identifier	6	254*
8	Count of external identifiers of one translation unit	511	∞
9	Count of identifiers having the block valid range in one block	127	∞
10	Count of macro names that can be simultaneously defined by one translation unit	1024	∞
11	Count of virtual arguments in one function definition	31	∞
12	Count of arguments for one function call	31	∞
13	Count of virtual arguments in one macro definition	31	∞
14	Count of arguments in one macro call	31	∞
15	Maximum count of characters in one logical source line	509	∞
16	Count of characters in a (linked) byte character string literal or wide-angle character string literal (terminal character included)	509	∞
17	Count of bytes of one arithmetic unit	32767	65535
18	Count of nesting levels for #include file	8	252



Table 5.12-1 List of Translation Limitations (2 / 2)

No.	Function	ANSI Standard	Compiler
19	Count of case name cards in one switch statement (excluding nested switch statements)	257	∞
20	Count of members of one structure or union	127	∞
21	Count of enumerated type constants in one enumerated type	127	∞
22	Count of structure or union nesting levels for one structure declaration array	15	∞

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 output to the assembler. If there are identifiers whose 254 leading characters are the same, an error may occur in the assembler.

CHAPTER 6

EXECUTION ENVIRONMENT

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

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

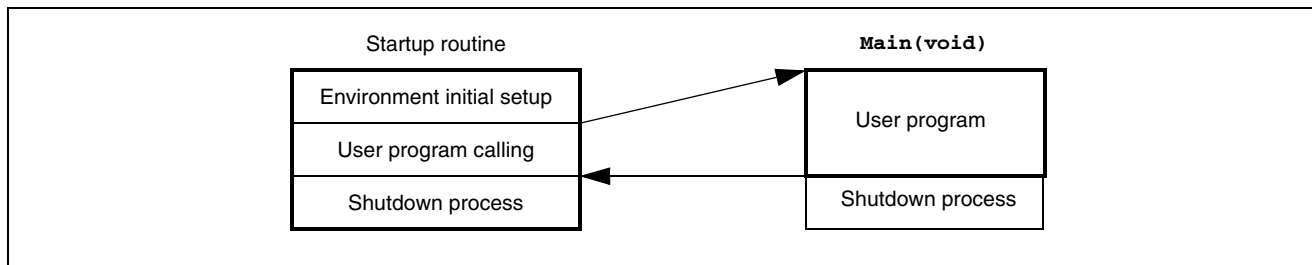
The void main(void), which is normally used as the function that the startup routine calls in the program start process, is to be called.

- 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



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

The compiler uses the general-purpose register. Therefore, it is necessary to specify the register bank that is used it by the startup routine and the linker option (-rg option).

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 “4.6.5 Register of fcc896s Command”. 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 fcc896s Command Register Status Prevailing at Return from User Program

Register	Value Guarantee at Return
A, T and EP	Not provided
R0 and R1	Not provided
R2 to R7	Provided
IX	Provided
SP	Provided

6.2 Startup Routine Creation

This section describes the processes necessary for startup routine creation.

■ Startup Routine Creation

1. Register Initial Setup

Set the stack pointer (SP) to the top of the stack (stack top).

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 and DIRDATA sections to 0.

3. Initialization Data Area Duplication

When incorporating constant data or program into ROM, the data positioned in the ROM area needs to be copied to the RAM area. However, this duplication step is unnecessary if such a data rewrite operation will not be performed within the user program.

The area to be incorporated into ROM is usually positioned in the INIT section. When incorporation into ROM is specified, the linker automatically generates the following symbols for the specified section name.

ROM_ Specified section name

RAM_ Specified section name

The above symbols indicate the ROM and RAM area start addresses, respectively. An example specifying of incorporation into ROM for the INIT section is shown below.

```
> fcc896s -ro ROM=ROM Address range -ra RAM=RAM Address range -SC @INIT=ROM, INIT=RAM ...
```

For the details of incorporation into ROM, refer to the Linkage Kit Manual.

4. User Program Calling

Call the user program.

5. 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.

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

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 of fcc896s Command

Four fcc896s command library files are provided below.

lib896.lib (General-purpose standard library for F²MC-8L)

lib950.lib (General-purpose standard library for F²MC-8FX)

lib896if.lib (Simulator debugger low-level function library for F²MC-8L)

lib950if.lib (Simulator debugger low-level function library for F²MC-8FX)

● Header Files

The header files are shown below.

assert.h

cctype.h

errno.h

float.h

limits.h

math.h

setjmp.h

stdarg.h

stddef.h

stdio.h*

stdlib.h

string.h

* The file operation function is not supported.

■ Library Section Names

Table 7.1-1 shows the section names used by the libraries.

Table 7.1-1 fcc896s Command Library Section Names

Section Type	Section Name
Code section	CODE
Data section	DATA
Initialized section	INIT
Constant section	CONST

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 “8.4 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 Low-level Function Types
- 8.3 Standard Library Functions and Required Process/Low-Level Functions
- 8.4 Low-Level Function Specifications

8.1 Library Incorporation Overview

This section outlines library incorporation.

■ Processes and Functions Required for Library Use

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 the standard library, and whenever such processes are needed, they will be called as a low-level function.

The following processes must be prepared for library use.

- Low-level function creation

At the time of library incorporation, the above processes and functions must be prepared in accordance with the system.

8.2 Low-level Function Types

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

- 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 exercise program execution control.

■ Low-level Function Types

- 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.

8.3 Standard Library Functions and Required Process/Low-Level Functions

This section describes the standard library functions and associated low-level functions.

■ Standard Library Functions and Required Process/Low-level Functions

Table 8.3-1 shows the relationship between the standard library functions that use the low-level functions and the associated low-level functions.

Table 8.3-1 Standard Library Functions and Required Low-level Functions

Standard Library Function	Low-level Function	
abort ()		_abort ()
calloc () malloc () realloc () free ()	sbrk ()	
exit ()		_exit ()

8.4 Low-Level Function Specifications

There are various low-level functions. 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.

8.4.1 sbrk Function

Create the `sbrk` function in compliance with the specifications stated in this section.
`char *sbrk( INT SIZE);`

■ sbrk Function

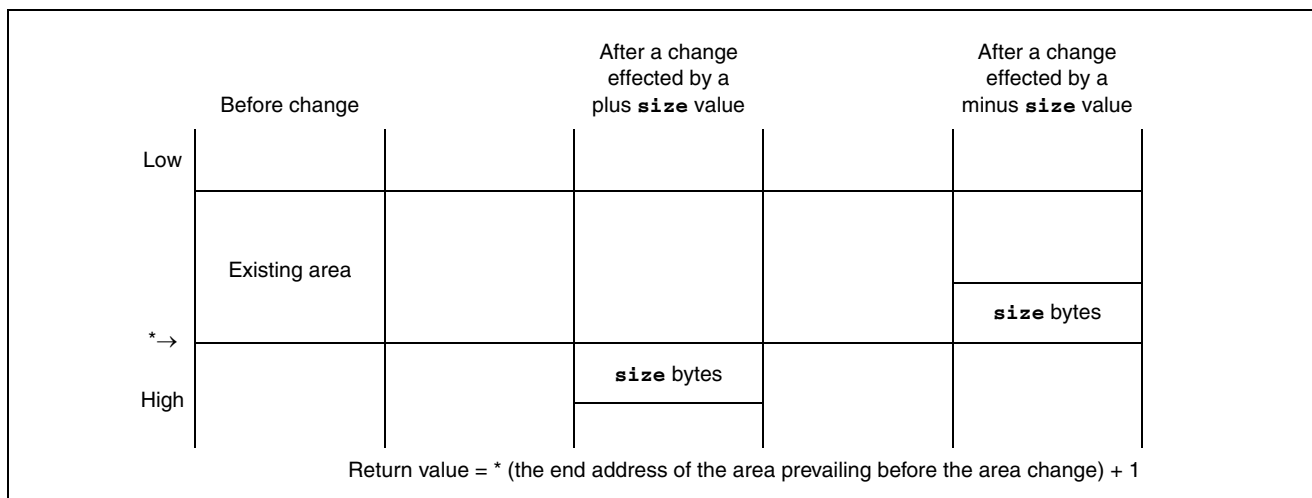
[Explanation]

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 8.4-1 Area Change Brought About by sbrk Function Calling



[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.

8.4.2 `_exit` Function

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

```
#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.

8.4.3 `_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 Data 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 9.1-1 Compiler-dependent Language Specification Differentials (1 / 2)

Specification Differentials	Associated JIS standard	Compiler
Japanese language process support and code system	5.2.1 Character Set 6.1.2 Identifier	No support EUC and SJIS entries can be made only in the comment.
Recognized character count of an identifier with an external binding	6.1.2 Identifier	254 leading characters
Differentiation between upper- and lower-case alphabetical characters of an identifier with an external binding	6.1.2 Identifier	Treated as different characters
Character set element expression code system	6.1.3 Constant	ASCII code
Char type treatment and expressible value range	6.2.1.1 Character Type and Integer Type	Unsigned* ¹ 0 to 255
Floating-point data formats and sizes float type double/long double types	6.1.2.5 Type	IEEE type* ² 4 bytes 8 bytes
Whether or not to treat the start bit as signed bit when following types are specified as bit field char, short int, int, and long int types	6.5.2.1 Structure Specifier and Union Specifier	Not treated as a sign* ¹
Types that can be specified as bit field	6.5.2.1 Structure Specifier and Union Specifier	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

Table 9.1-1 Compiler-dependent Language Specification Differentials (2 / 2)

Specification Differentials	Associated JIS standard	Compiler
Structure or union type member boundary alignment value 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 float type double type long double type Pointer type	6.5.2.1 Structure Specifier and Union Specifier	1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte 1 byte
Character constant expression code system for preprocessor	6.8.1 Conditional Acquisition	ASCII code
Registers that can be specified within asm statement		A, AL, and AH* ³
ANSI-compliant standard library function support		Refer to the volume entitled <i>Libraries</i> .

*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 9.2-1 Floating-point Data Format and Expressible Value Range

Floating-point Data Format	Expressible Value Range
float type	The exponent part is a value between $2 - 126$ and $2 + 127$. The fractional portion of the mantissa (the integer portion is normalized to 1) is binary and has 24-digit accuracy.
double type	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.
long double type	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.

9.3 Floating-Point Data Operation due to the Runtime Library Function

All floating-point data 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-zero mode, only.

- Precision of division

The least significant bit of the resultant mantissa part might contain an error.

- 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 sign of addition and subtraction

If both operands are zero, the sign of the returned value is uncertain.

- Resultant sign of multiplication

If either of operands is zero, the sign of the returned value is uncertain.

- Resultant value of division

If both operands are zero, zero with the uncertain sign is returned. If only the right operand is zero, the uncertain value is returned.

- Resultant operation 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, uncertain value is returned.

- Resultant operation under the overflow exception

Uncertain value is returned.

- Resultant operation under the invalid operation exception

Uncertain value is returned.

- Interrupt at operation exception

No interrupt occur.

- Status flag

Not supported.

■ Comparison

- Denormalized number

Comparing between two denormalized numbers or between zero and a denormalized number, the library function returns uncertain result.

- Operation result under the invalid operation exception

The library function returns uncertain result.

- 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 operation 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 operation sign

If the converting value is zero, zero with positive sign is always returned.

- Resultant operation 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, uncertain value is returned.

- Resultant operation under the overflow exception

Uncertain value is returned.

- Resultant operation under the invalid operation exception

Uncertain value is returned.

- 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 fcc896s Command 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.

- Dynamic memory allocation function (sbrk)

In the simulator debugger, the program executed cannot terminate its own execution. Therefore, prepare the `_abort` and `_exit` functions.

■ 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 fcc896s Command Low-Level Function Library Use

This section describes the load module creation for low-level function library use.

■ About the Low-level Library for the Simulator Debugger in the fcc896s Command

In the library of the fcc896s command, the file operation function is not supported. Therefore, only the sbrk function is registered in low level library lib896if.lib for the simulator debugger.

■ Load Module Creation

Please compile and link all necessary modules after making a necessary program. It is not necessary to specify a special option. The following library and the startup routine are linked.

- startup.obj
- Standard library (lib896.lib)
- Low-level function library (lib896if.lib)

■ Example

The program which secures the area in 100 bytes by the malloc function is made and executes the simulator debugger.

```
#include <stdlib.h>
main()
{
    void *ptr;
    ptr = malloc(100);
}
```

Create a C-source file named test.c as shown above.

Compile using the following command. Setup the corresponding directory for LIBTOOL.

```
fcc896s test.c -cpu MB89P935B -L LIBTOOL -l lib896.lib
-l lib896if.lib
```

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.

```
> go , end
```



10.3 Low-Level Func. Function

This section describes the function specific to the simulator debugger low-level functions.

■ 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".

10.4 Low-Level Function Library Change

This section describes how to change the dynamic allocation area.

■ fcc896s Command 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 HEAP_SIZE 5*1024
```

Use the following commands to compile and update the library.

```
> fcc896s -O -c sbrk.c -cpu MB89P935B
```

```
> flib896s -r sbrk.obj lib896if.lib -cpu MB89P935B
```

When the above change is made, the dynamic allocation area is secured as the sbrk.c static external variable without being positioned at the beginning of the stack.

■ fcc896s Command 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 HEAP_SIZE      5*1024
static long brk_siz = 0;
static char _heap[HEAP_SIZE];
#define _heap_size HEAP_SIZE
extern char *sbrk(int size)
{
    if (brk_siz + size > _heap_size || brk_siz + size < 0)
        return ((char *)-1);
    brk_siz += size;
    return (_heap + 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).

ANSI standard non-conforming specifications are described (C).

Reentrancy of the C library functions are described (D).

The list of the error messages are described (E).

APPENDIX A List of Type, Macro, Variable, and Function

APPENDIX B Operations Specific to Libraries

APPENDIX C ANSI Standard Non-Conforming Specifications

APPENDIX D About Reentrancy of the C Library Functions

APPENDIX E Error Message

APPENDIX F Major Changes

APPENDIX A List of Type, Macro, Variable, and Function

This section lists the types, macros, variables, and functions provided by the libraries.

■ ctype.h

● Macros

isalnum	isalpha	iscentrl	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_DIG	DBL_DIG	LDBL_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				

■ 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

● Macros

BUFSIZ

● Function

sprintf sscanf vsprintf

■ 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
strtoul	rand	srand	calloc	free
malloc	realloc	abort	atexit	exit
bsearch	qsort	abs	div	labs
ldiv				

■ string.h

- Type

ptrdiff_t	size_t
-----------	--------

- Macros

NULL	offsetof
------	----------

- Function

memcpy	memmove	strcpy	strncpy	strcat
strncat	memcmp	strcmp	strncmp	memchr
strchr	strcspn	strpbrk	strchr	strspn
strstr	strtok	memset	strlen	

■ setjmp.h

- Type

jmp_buf

- Macros

setjmp

- Function

longjmp

APPENDIX B Operations Specific to Libraries

This section describes the operations specific to the libraries.

■ Operations Specific to Libraries

(1) 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: \100 to \037, or \177
- islower: a to z
- isprint: \040 to \176
- isupper: A to Z

(2) Mathematical function return value upon definition area error occurrence

- qNaN

(3) 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.

(4) 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.

(5) %p format conversion input format for sscanf function

The sscanf function adds leading 0s if the digit count is less than 4 when using the upper- or lower-case alphabetic character-based hexadecimal notation. If the specified digit count (4 digits) is exceeded, only the lower-order part is valid.

(6) 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].
```

(7) 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.

(8) Floating-point number limit values

- FLT_MAX7F7F FFFF
- DBL_MAX7FEF FFFF FFFF FFFF
- FLT_EPSILON3400 0000
- DBL_EPSILON3CB0 0000 0000 0000

- FLT_MIN0080 0000
- DBL_MIN0010 0000 0000 0000

(9) 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.

(10) Limitations on va_start macro

Do not use the following variable definitions for **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
- Type different from the type derived from existing argument extension

(11) div_t type and ldiv_t type

It is equivalent to an undermentioned structure.

```
div_t:struct {
    int  quot;
    int  rem;
};

ldiv_t:struct {
    long int  quot;
    long int  rem;
};
```

(12) abort function operations

When the abort function is called, the _abort function is called.

(13) Maximum count of functions that can be registered by the atexit function

Up to 32 functions can be registered.

(14) 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.

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.

APPENDIX C ANSI Standard Non-Conforming Specifications

The ANSI standard non-conforming specifications are described.

To generate the instructions of 8-bit microcontroller effectively, the following ANSI standard non-conforming specifications were supported.

These specifications enable the reduction of the code size and the improvement of the execution speed.

- The integral promotions are not applied.
- The type of the integer constant has a minimum type.
- The type of the enumeration type has a minimum type.
- The type of the truth-value of the operation result has type char.
- The default argument promotions are not applied.

When the -Jn option is specified, these specifications are effective.

The behavior of the program might vary because of the ANSI non-standard conforming specification.

Please confirm the caution for each specification.

■ The Integral Promotions are not Applied

[Explanation]

The reduction in the type conversion code and the generation of instructions for 1-byte type operation are promoted by not applying the integral promotions.

When the integral promotions are not applied, the rule of the usual arithmetic conversions are as follows.

- First, if either operand has type long double, the other operand is converted to long double.
- Otherwise, if either operand has type double, the other operand is converted to double.
- Otherwise, if either operand has type float, the other operand is converted to float.
- Otherwise, if either operand has type unsigned long int, the other operand is converted to unsigned long int.
- Otherwise, if either operand has type long int, the other operand is converted to long int.
- Otherwise, if either operand has type unsigned int, the other operand is converted to unsigned int.
- Otherwise, if either operand has type int and the other operand has type unsigned short int, both operands converted to unsigned int.
- Otherwise, if either operand has type int, the other operand is converted to int.
- Otherwise, if either operand has type unsigned short int, the other operand is converted to unsigned short int.
- Otherwise, if either operand has type short int, the other operand is converted to short int.
- Otherwise, if either operand has type unsigned char, the other operand is converted to unsigned char.
- Otherwise, if either operand has type char, the other operand is converted to char.
- Otherwise, both operands have type signed char.

Note:

When the integral promotions are not applied, it is necessary to consider the overflow generated by 1-byte type operation.

■ The Type of the Integer Constant has a Minimum Type**[Explanation]**

When integer constant, enumeration constant and character constant are represented by signed char type or unsigned char type, these constants are treated as 1-byte type.

As a result, the generation of instructions for 1-byte type operation is promoted.

Notes:

- When the integer constant has a minimum type, it is necessary to consider the overflow generated by 1-byte type operation.
 - When an integer constant represented by signed char type or unsigned char type is described in the argument of the function to which the prototype is not declared, the value of the higher byte of 1-byte argument is irregular. To evade this issue, it is necessary that argument cast to an original type.
 - When unary-minus operator is applied to constant C that is represented by unsigned char type (1U to 127U and 128 to 255), value of result is "256 - C". To evade this issue, it is necessary to add suffix 'L' to the constant C.
-

■ The Type of the Enumeration Type has a Minimum Type**[Explanation]**

The enumeration type has a minimum type that represents the value of the enumeration constant.

As a result, the variable area of the enumeration type is reduced and the generation of instructions for 1-byte type operation is promoted.

Note:

The assignment of the variable between different enumeration types might not be operated correctly.

■ The Type of the Truth-value of the Operation Result has Type Char

[Explanation]

The generation of instructions for 1-byte type operation is promoted by changing the type of the truth-value of the operation result into the type char.

Notes:

- When the type of the truth-value of the operation result has a char type, it is necessary to consider the overflow generated by 1-byte type operation.
 - When the truth-value of the operation result is used in the argument of a function to which the prototype is not declared, the value of the higher byte is irregular. To evade this issue, it is necessary that argument is cast to the original type.
-

■ The Default Argument Promotions are not Applied

[Explanation]

The type conversion code for argument is deleted by not applying the default argument promotions.

Note:

When the default argument promotions are not applied, the value of the higher byte of 1-byte type argument is irregular. To evade this issue, it is necessary that the argument is cast to the original type.

APPENDIX D About Reentrancy of the C Library Functions

Reentrancy of the C library functions are described.

■ About Reentrancy of the C Library Functions

- List of reentrant functions

Table D-1 List of Reentrant Functions

Function name	Header file name	Function name	Header file name
abs	stdlib.h	memmove	string.h
atoi	stdlib.h	memset	string.h
atol	stdlib.h	strcat	string.h
bsearch	stdlib.h	strchr	string.h
div	stdlib.h	strcmp	string.h
isalnum	ctype.h	strcpy	string.h
isalpha	ctype.h	strcspn	string.h
isctrl	ctype.h	strlen	string.h
isdigit	ctype.h	strncat	string.h
isgraph	ctype.h	strncmp	string.h
islower	ctype.h	strncpy	string.h
isprint	ctype.h	strpbrk	string.h
ispunct	ctype.h	strrchr	string.h
isspace	ctype.h	strspn	string.h
isupper	ctype.h	strstr	string.h
isxdigit	ctype.h	tolower	ctype.h
labs	stdlib.h	toupper	ctype.h
ldiv	stdlib.h	va_arg	stdarg.h
memchr	string.h	va_end	stdarg.h
memcmp	string.h	va_start	stdarg.h
memcpy	string.h	All runtime library functions	

- List of non-reentrant functions

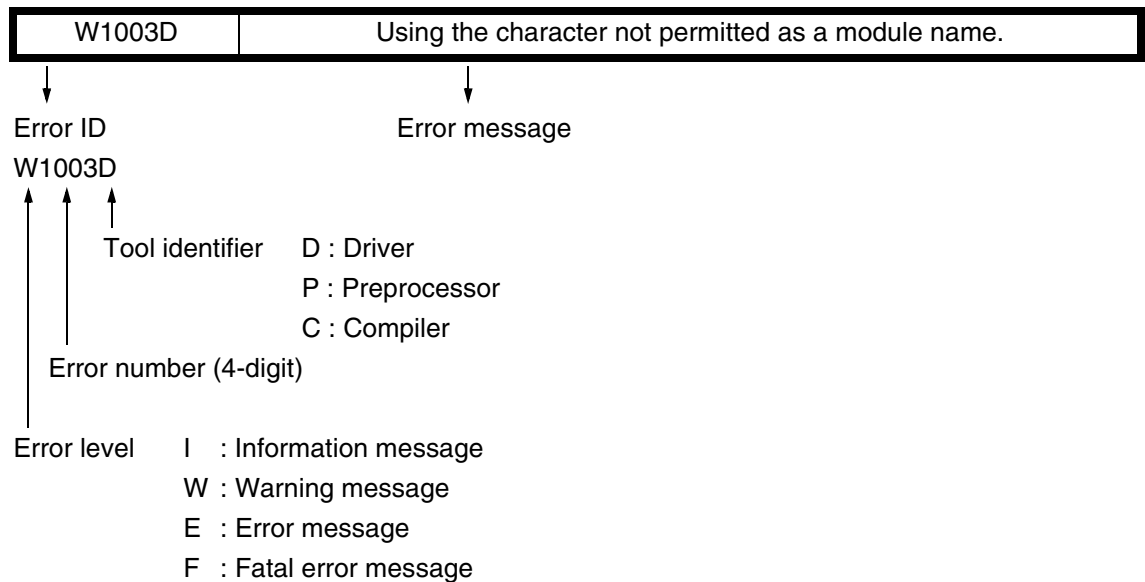
Table D-2 List of Non-reentrant Functions

Function name	Header file name	Function name	Header file name
acos	math.h	tanh	math.h
asin	math.h	longjmp	setjmp.h
atan	math.h	setjmp	setjmp.h
atan2	math.h	sprintf	stdio.h
ceil	math.h	sscanf	stdio.h
cos	math.h	vsprintf	stdio.h
cosh	math.h	abort	stdlib.h
exp	math.h	atexit	stdlib.h
fabs	math.h	atof	stdlib.h
floor	math.h	calloc	stdlib.h
fmod	math.h	exit	stdlib.h
frexp	math.h	free	stdlib.h
ldexp	math.h	malloc	stdlib.h
log	math.h	qsort	stdlib.h
log10	math.h	rand	stdlib.h
modf	math.h	realloc	stdlib.h
pow	math.h	srand	stdlib.h
sin	math.h	strtod	stdlib.h
sinh	math.h	strtol	stdlib.h
sqrt	math.h	strtoul	stdlib.h
tan	math.h	strtok	string.h

APPENDIX E Error Message

The compiler displays the error message below.

■ Format of Error Message



W1003D	Using the character not permitted as a module name.
--------	---

[Explanation]

Characters other than alphabet (A to Za to z), numeric character (0 to 9), and underscore (_) are used as module name.

W1005D	invalid option(option_name)
--------	-----------------------------

[Explanation]

An invalid option was specified. Please refer to the explanation concerning the option to confirm it.

E4002D	Illegal option-name
--------	---------------------

[Explanation]

The option name has some errors.

E4003D	Illegal value
--------	---------------

[Explanation]

The sub-option value is incorrect.

E4004D	Illegal sub-option
--------	--------------------

[Explanation]

The sub-option has some errors.

E4005D	Illegal parameter description
--------	-------------------------------

[Explanation]

The option specification has some errors.

E4006D	Can not open option-file
--------	--------------------------

[Explanation]

The option file cannot be opened.

E4007D	Nested option-file exceeds 8
--------	------------------------------

[Explanation]

The nest level of option-file exceeds 8.

E4008D	Insufficient memory
--------	---------------------

[Explanation]

The memory capacity is insufficient.

E4009D	Illegal file name
--------	-------------------

[Explanation]

The file name has some errors.

E4010D	internal error : Illegal reserve number
--------	---

[Explanation]

Internal error : the sub-option reserved number is illegal.

E4011D	Illegal section information
--------	-----------------------------

[Explanation]

The section information has some errors.

E4012D	Illegal tool-item
--------	-------------------

[Explanation]

The tool-item has some errors.

E4013D	Illegal sub-option
--------	--------------------

[Explanation]

The language specification has some errors.

E4014D	Illegal optimize level
--------	------------------------

[Explanation]

The optimize specification has some errors..

E4015D	internal error : Illegal prefix
--------	---------------------------------

[Explanation]

Internal error: the work file prefix is illegal.

E4016D	sub process call is failed
--------	----------------------------

[Explanation]

The activation of sub process is failed.

E4017D	tool execute is failed
--------	------------------------

[Explanation]

The tool calling is failed.

E4018D	help-file is not found
--------	------------------------

[Explanation]

The help-file is not found.

E4019D	can't unlink the file
--------	-----------------------

[Explanation]

The file cannot be deleted.

E4020D	cannot process the C++ source file
--------	------------------------------------

[Explanation]

The C++ source file cannot be processed.

E4021D	option -cpu is not defined.
--------	-----------------------------

[Explanation]

The option -cpu is not specified.

E4022D	CPU information file not found
--------	--------------------------------

[Explanation]

The CPU information file is not found.

E4023D	CPU information not found
--------	---------------------------

[Explanation]

The CPU information is not found.

E4024D	too long file name.
--------	---------------------

[Explanation]

The file name is too long.

E4025D	Can not open file
--------	-------------------

[Explanation]

The file cannot be opened.

F9001D	internal error : can not find file
--------	------------------------------------

[Explanation]

Internal error: the file is not found.

F9002D	internal error : Illegal error-no
--------	-----------------------------------

[Explanation]

Internal error: the error message number is illegal.

F9003D	Cannot create Ctrl-C Thread
--------	-----------------------------

[Explanation]

The Ctrl-C thread cannot be created.

F9004D	Internal error
--------	----------------

[Explanation]

A program internal error is generated.

F9005D	Insufficient memory
--------	---------------------

[Explanation]

The memory capacity is insufficient.

F9006D	Illegal CPU information file format
--------	-------------------------------------

[Explanation]

The CPU information file format is incorrect.

W1001P	##R: empty expression: identifier required
--------	--

[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.

W1002P	##R: invalid token specified after identifier: newline expected
--------	---

[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.

W1003P	##R: expression expected
--------	--------------------------

[Explanation]

No constant expression to evaluate is specified to the #if or #elif directive. Continues the compilation assuming that 0 is specified.

W1004P	##R: digit-sequence expected
--------	------------------------------

[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.

W1005P	too many arguments %d for macro '%s': %d expected
--------	---

[Explanation]

The number of arguments specified in the macro expansion exceeds that of parameters in the definition. The excess is ignored.

W1006P	too few arguments %d for macro '%s': %d expected
--------	--

[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.
Remark: 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.

Remark: 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%
--------	-----------------------

[Explanation]

The character is internal use only in the preprocess. Continues the compilation assuming it as a white space.

W1024P	comment replaced with `##' operator
--------	-------------------------------------

[Explanation]

An old style specification is applied which regards "a/**/b" as "a##b". Continues the compilation making it valid.

W1025P	##R: cannot define macro `%'
--------	------------------------------

[Explanation]

The identifier is reserved by the preprocessor and user cannot redefine it as macro name. Continues the compilation making the macro definition invalid.

W1026P	##R: macro `%' redefined
--------	--------------------------

[Explanation]

The identifier is already defined as macro differently. Continues the compilation making the preceding definition valid.

W1027P	##R: invalid token specified after identifier: newline expected
--------	---

[Explanation]

An excess token is found after the identifier in the #undef directive. Continues the compilation neglecting the token.

W1028P	##R: cannot undefine macro '%s'
--------	---------------------------------

[Explanation]

The macro name specified in identifier of the #undef directive cannot be canceled. Continues the compilation making the #undef directive invalid.

W1029P	invalid token after #else
--------	---------------------------

[Explanation]

An excess token is found in the #else directive. Continues the compilation neglecting it.

W1030P	invalid token after #endif
--------	----------------------------

[Explanation]

An excess token is found in the #endif directive. Continues the compilation neglecting it.

W1031P	invalid token follows at the end of #include directive line
--------	---

[Explanation]

An excess token is found after the file name in the #include directive. Continues the compilation neglecting it.

W1032P	#include: cannot find file \"%s\"
--------	-----------------------------------

[Explanation]

The private include file is not found.
Remark: This message is not used.

W1033P	#include: cannot find file <%s>
--------	---------------------------------

[Explanation]

The standard include file is not found.

Remark: This message is not used.

W1034P	##R: syntax error: identifier expected
--------	--

[Explanation]

No identifier is specified in the #assert or #unassert directive. Continues the compilation neglecting the directive.

Remark: This message is not used.

W1035P	##R: '(' required
--------	-------------------

[Explanation]

The '(' is needed after an identifier in the #assert or #unassert directive. Continues the compilation neglecting the directive.

W1036P	##R: syntax error: `,' or `)' expected
--------	--

[Explanation]

The `,' or `)' is needed to separate or to end the assertion specifier list in the #assert or #unassert directive. Continues the compilation neglecting the directive.

Remark: This message is not used.

W1037P	##R: newline expected: invalid token follows after `)'
--------	--

[Explanation]

An excess token is found after the `)' which encloses assertion specifiers in #assert or #unassert directive. Continues the compilation neglecting the token.

W1038P	##R: filename expected
--------	------------------------

[Explanation]

An excess token is found after the line number of the #line directive besides the file name character string. Continues the compilation assuming that no file name is specified.

W1039P	##R: line number 0 specified
--------	------------------------------

[Explanation]

The line number specified in the #line directive is 0. Continues the compilation neglecting the #line directive.

W1040P	##R: specified line number is greater than 32767
--------	--

[Explanation]

According to the ANSI standard, the line number in the #line directive should be equal to or less than 32767. Continues the compilation making the specified value valid.

W1041P	macro '%s' recursion
--------	----------------------

[Explanation]

Macro replacement is evaluated recursively. Continues the compilation making the evaluation valid.

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.

Remark: 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.

W1048P	unknown option %s
--------	-------------------

[Explanation]

Unknown option is detected.
Remark: This message is not used.

W1049P	invalid option %s
--------	-------------------

[Explanation]

Unknown option is detected.
Remark: 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.

Remark: 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.

Remark: 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.

Remark: This message is not used.

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.

Remark: 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 <code>\\%c'</code>
--------	--

[Explanation]

A character which not specified as an escape sequence in the ANSI C is found after the `\`. Continues the compilation neglecting the `\`.

W1065P	too large integer constant for radix <code>%d</code>
--------	--

[Explanation]

The digit-sequence is too long and surely overflow before converting into the internal format. Continues the compilation assuming that 0 is specified.

W1066P	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.

W1067P	escape sequence does not fit in range of wide character
--------	---

[Explanation]

The result of the escape sequence cannot be represented by four bytes. Continues the compilation making only the lowest wide character valid.

Remark: This message is not used.

W1068P	too large integer constant <code>`%s'</code> for radix <code>%d</code>
--------	--

[Explanation]

The digit-sequence is too long and surely overflow before converting into the internal format. Continues the compilation assuming that 0 is specified.

W1069P	exceed the maximum length of octal escape sequence
--------	--

[Explanation]

The octal number written in the escape sequence form consists of more than four digits.

Remark: This message is not used.

W1070P	unknown escape sequence '\\%o'
--------	--------------------------------

[Explanation]

A character which is not specified as an escape sequence in the ANSI C is found after the '\\'. Continues the compilation neglecting the '\\'.

W1071P	too long multi-character character constant
--------	---

[Explanation]

A character constant which consists of more than five characters is found. Continues the compilation making the first four characters valid.

W1072P	multi-character character constant specified
--------	--

[Explanation]

A character constant which consists of more than two characters is found. Continues the compilation making it valid.

W1073P	cannot open compiler message file \"%s\"
--------	--

[Explanation]

A message built in the preprocessor is issued.

W1074P	assertion is ANSI C extension
--------	-------------------------------

[Explanation]

The assertion is not the ANSI C standard. Continues the compilation evaluating it as an assertion.

W1075P	##R: invalid token: identifier required
--------	---

[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.

W1076P	##R: too large decimal constant
--------	---------------------------------

[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.

W1077P	##operator used in macro replacement of non function-like macro
--------	---

[Explanation]

The function of the ``#` operator is converting parameters of a function-like macro into string literals.
It replaces as normal ``#` character.

W1078P	unterminated string literal: EOF in string literal
--------	--

[Explanation]

The last double quotation is missing to enclose the string literal.
Remark: This message is not used.

W1079P	unterminated character constant: EOF in character constant
--------	--

[Explanation]

The last single quotation is missing to enclose the character constant.
Remark: This message is not used.

W1080P	unterminated string literal: newline in string literal
--------	--

[Explanation]

The last double quotation is missing to enclose the string literal.
Remark: This message is not used.

W1081P	unterminated character constant: newline in character constant
--------	--

[Explanation]

The last single quotation is missing to enclose the character constant.

Remark: This message is not used.

W1082P	too long identifier, truncated to '%s'
--------	--

[Explanation]

The identifier is too long.

Remark: 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.

Remark: 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.

Remark: This message is not used.

W1085P	decimal integer constant is too large
--------	---------------------------------------

[Explanation]

The decimal integer constant is too large.

Remark: 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.

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.

E4010P	assertion: '(' required after '#identifier'
--------	---

[Explanation]

The '(' is missing to enclose the assertion specifier in referring an assertion predicate name.

E4011P	assertion: empty within parentheses
--------	-------------------------------------

[Explanation]

No identifier is specified as an assertion specifier in referring an assertion predicate name.

E4012P	identifier required after 'defined' operator
--------	--

[Explanation]

No identifier is specified after the defined operator.

Remark: This message is not used.

E4013P	assertion: ')' required after '#identifier(...'
--------	---

[Explanation]

The ')' is missing to enclose the assertion specifier in referring an assertion predicate name.

E4014P	write error (fwrite())
--------	------------------------

[Explanation]

An error occurred while writing the output of the preprocessor to a file.

E4015P	##R: duplicate formal parameter
--------	---------------------------------

[Explanation]

Some parameters specified in the function-like macro definition by the #define directive have the same spelling.

E4016P	##R: parameter syntax error: ', ' or ')' expected
--------	---

[Explanation]

In the function-like macro definition by the #define directive, a separator ', ' or a terminator ')' is missing in the parameter list.

E4017P	##R: invalid parameter: identifier required
--------	---

[Explanation]

In the function-like macro definition by the #define directive, a parameter which is not an identifier is specified.

E4018P	##R: invalid token: identifier required
--------	---

[Explanation]

The macro name of a #define or #undef directive or a #assert predicate name should be an identifier.

E4019P	##R: empty macro name: identifier required
--------	--

[Explanation]

No identifier is specified as a macro name in the #define or #undef directive.

E4020P	unexpected #elif
--------	------------------

[Explanation]

The #if for the #elif directive is missing.

E4021P	#elif follows after #else
--------	---------------------------

[Explanation]

The #elif directive appears after the #else directive. The #else directive should appear after the #elif directive.

E4022P	unexpected #else
--------	------------------

[Explanation]

The #if for the #else directive is missing.

E4023P	#else follows #else
--------	---------------------

[Explanation]

The #else directive appears after the #else directive. In a #if closes, the #else directive should appear only once.

E4024P	unexpected #endif
--------	-------------------

[Explanation]

The #if for the #endif directive is missing.

E4025P	#include: empty directive line
--------	--------------------------------

[Explanation]

No file name is specified in the #include directive.

E4026P	#include: invalid character: '<' or '\"' expected
--------	---

[Explanation]

The file name in the #include directive does not begin with '<' nor '\"'.

E4027P	#include: unterminated filename: '%c' expected
--------	--

[Explanation]

The file name in the #include directive does not end with '% ' nor '\"'.

E4028P	#include: empty filename
--------	--------------------------

[Explanation]

No file name is specified within <> or '\"' in the #include directive.

E4029P	##R: digit-sequence expected
--------	------------------------------

[Explanation]

No digit-sequence is specified for line number in the #line directive.

E4030P	##R: too large decimal constant
--------	---------------------------------

[Explanation]

The decimal digit-sequence for line number in the #line directive is too long.

E4031P	#%R: \" missing
--------	-----------------

[Explanation]

The character string expressed the file name specified in the #line directive does not end with \".

E4032P	defined operator: invalid token: identifier expected
--------	--

[Explanation]

The operand within the parenthesis in the defined operator is not an identifier.

E4033P	defined operator: `)' expected
--------	--------------------------------

[Explanation]

Though the operand of the defined operator begins with `(', it does not end with `)'.

E4034P	defined operator: identifier or `(' expected
--------	--

[Explanation]

A token other than identifier nor `(' appears just after the defined operator.

E4035P	unterminated macro `%s': `)' expected
--------	---------------------------------------

[Explanation]

The `)' is missing to enclose arguments in the function-like macro reference.

E4036P	unterminated character constant: `\" expected
--------	---

[Explanation]

The single quotation is missing to terminate the character constant.

Remark: This message is not used.

E4037P	unterminated string literal: \"\" expected
--------	--

[Explanation]

The double quotation is missing to terminate the string literal.

Remark: This message is not used.

E4038P	#include: cannot find file \"%s\"
--------	-----------------------------------

[Explanation]

The private include file is not found.

E4039P	#include: cannot find file <%s>
--------	---------------------------------

[Explanation]

The standard include file is not found.

E4040P	%%R: syntax error: identifier expected
--------	--

[Explanation]

No identifier is specified for a parameter between `,' and `)' in the function-like macro definition by #define directive.

E4041P	%%R: empty in parentheses
--------	---------------------------

[Explanation]

No identifier is written within the parenthesis enclosing an assertion specifier in the #assert or #unassert directive.

E4042P	unknown directive
--------	-------------------

[Explanation]

Unknown preprocessor directive is specified.

E4043P	%%R: syntax error: `)' expected
--------	---------------------------------

[Explanation]

The `)' is missing to enclose the assertion specifier in the #assert or #unassert directive.

E4044P	%%R: token sequence expected
--------	------------------------------

[Explanation]

Nothing is written after the `(' which leads to an assertion specifier in the #assert or #unassert directive.

E4045P	too nested include file
--------	-------------------------

[Explanation]

The nest level of include exceeds the limit of this system, 64.

E4046P	unterminated string literal: `\" expected
--------	---

[Explanation]

The `\" is missing to terminate the string literal.

Remark: This message is not used.

E4047P	EOF in comment
--------	----------------

[Explanation]

The "*/" is missing to terminate the comment written in the condition expression.

E4048P	EOF in string literal
--------	-----------------------

[Explanation]

The `\" is missing to terminate the string literal in the condition expression.

E4049P	sorry, internal limitation: quoted character too long
--------	---

[Explanation]

The length of the character constant written in condition expression or the hexadecimal beginning with "x" in the string literal exceeds the limit of this system, 4028.

E4050P	character constant has no character expression
--------	--

[Explanation]

No character is written in the character constant in the condition expression.

E4051P	unterminated character constant: newline in character constant
--------	--

[Explanation]

Newline is found in the character constant.

Remark: This message is not used.

E4052P	unterminated character constant: EOF in character constant
--------	--

[Explanation]

The single quotation is missing to terminate the character constant.

Remark: This message is not used.

E4053P	too many postfix characters '%c' for constant
--------	---

[Explanation]

The postfix of the numerical constant in the condition expression is not specified correctly.

E4054P	numeric octal constant contains invalid character
--------	---

[Explanation]

A character which is not an octal digit is used in the octal constant.

Remark: This message is not used.

E4055P	binary constant cannot be floating point constant
--------	---

[Explanation]

Binary constant beginning with '0b' or '0B' cannot be floating point constant.

Remark: This message is not used.

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.

E4062P	integer constant out of range
--------	-------------------------------

[Explanation]

The integral constant cannot be expressed internally.
Remark: This message is not used.

E4063P	invalid character `'%c'`
--------	--------------------------

[Explanation]

An invalid character is found in the condition expression.

E4064P	invalid binary constant
--------	-------------------------

[Explanation]

The first digit of the binary constant beginning with `0b' or `0B' is not 0 nor 1.

Remark: This message is not used.

E4065P	invalid hexadecimal constant
--------	------------------------------

[Explanation]

The first digit of the hexadecimal constant beginning with `0x' or `0X' is not a hexadecimal digit.

E4066P	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 in the condition expression.

E4067P	invalid multibyte string literal
--------	----------------------------------

[Explanation]

There is an invalid multi-byte character which cannot be recognized as a character code in the wide-character string literal.

Remark: This message is not used.

E4068P	invalid character `\\%o' (octal)
--------	----------------------------------

[Explanation]

An invalid character is found in the condition expression.

E4069P	%s near wide character string constant
--------	--

[Explanation]

This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the wide-character string literal.

E4070P	%s near wide character constant
--------	---------------------------------

[Explanation]

This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the wide-character constant.

E4071P	%s near '%s'
--------	--------------

[Explanation]

This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the specified character.

E4072P	%s detected
--------	-------------

[Explanation]

This is a supplemental message for syntax error in a condition expression: A syntax error occurred.

E4073P	%s near character constant
--------	----------------------------

[Explanation]

This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the character constant.

E4074P	%s near string constant
--------	-------------------------

[Explanation]

This is a supplemental message for syntax error in a condition expression: A syntax error occurred near the string literal.

E4075P	cannot quote EOF
--------	------------------

[Explanation]

The EOF is found just after the backslash.
Remark: This message is not used.

E4076P	invalid `\\` in input
--------	-----------------------

[Explanation]

The backslash is found in the condition expression.

E4077P	<<note:#error directive displays following literals written in the source file as they are>>
--------	--

[Explanation]

The "#error" is recognized.

E4078P	-default filename expected
--------	----------------------------

[Explanation]

No sub-parameter is set to the "-default" option to specify a file name.

E4079P	-predefine filename expected
--------	------------------------------

[Explanation]

No sub-parameter is set to the "-predefine" option to specify a file name.

E4080P	invalid `\$\$` in input
--------	-------------------------

[Explanation]

An invalid character `\$\$` is found in the condition expression.

E4081P	invalid option %s
--------	-------------------

[Explanation]

No sub-parameter is set to the "-X" option to specify language level specification.

Remark: This message is not used.

E4082P	write error (timeout)
--------	-----------------------

[Explanation]

An error occurred while writing the output of the preprocessor to a file.

Remark: This message is not used.

E4083P	write error (select())
--------	------------------------

[Explanation]

An error occurred while writing the output of the preprocessor to a file.

Remark: This message is not used.

E4084P	write error (write())
--------	-----------------------

[Explanation]

An error occurred while writing the output of the preprocessor to a file.

Remark: This message is not used.

E4085P	unknown # directive
--------	---------------------

[Explanation]

An unknown preprocessor directive is specified.

Remark: This message is not used.

E4086P	invalid character '\%o' (octal) in # directive line
--------	---

[Explanation]

An unknown character is found in the preprocessor directive.

Remark: This message is not used.

E4087P	cannot open file \"%s\"
--------	-------------------------

[Explanation]

The file specified in the "-Hf" option cannot be opened for writing.

F9001P	sorry, internal limitation: too nested input
--------	--

[Explanation]

Too many changes are made on the source file by macro substitution to process.

F9002P	detected too many errors to terminate compilation
--------	---

[Explanation]

Too many errors are detected to continue.

Remark: This message is not used.

F9003P	Broken pipe
--------	-------------

[Explanation]

Pipe is broken.

Remark: This message is not used.

F9004P	too many comments
--------	-------------------

[Explanation]

The number of comment exceeds the limit of this system, 16777215.

F9005P	cannot get file mode (fstat)
--------	------------------------------

[Explanation]

Cannot get the file information to read.

F9006P	fread
--------	-------

[Explanation]

Cannot read a file correctly.

F9007P	invalid file mode
--------	-------------------

[Explanation]

The file to read is not a correct text file.

F9008P	illegal message file
--------	----------------------

[Explanation]

The message file is not correct.

F9009P	virtual memory exhausted
--------	--------------------------

[Explanation]

Failed to gain the working memory area.

I0001C	previous declaration of `"%D": \"%s\", line %d
--------	--

[Explanation]

A supplemental message about a redeclaration of a symbol. The previous declaration is at the indicated line.

I0002C	on empty parameter declaration of function declaration
--------	--

[Explanation]

An information for an incompatibility of a type after the default argument promotion. A supplemental message of W1063C.

I0003C	on formal parameter declaration of function definition
--------	--

[Explanation]

An information for an incompatibility of a type after the default argument promotion. A supplemental message of W1063C.

I0004C	using ellipsis terminator on empty parameter declaration
--------	--

[Explanation]

An information for an incompatibility of a type after the default argument promotion. A supplemental message of W1063C.

I0005C	incompatible types between `%T' and `%T'
--------	--

[Explanation]

An information for the incompatibility of the type.

Remark: This message is not used.

I0006C	#pragma echo is not available on your system
--------	--

[Explanation]

The "#pragma echo" is not supported.

I0007C	`%D' is builtin symbol: `%T'
--------	------------------------------

[Explanation]

A supplemental message about the redeclaration of a symbol. The previous symbol is a built-in symbol.

W1006C	pointer to function specified
--------	-------------------------------

[Explanation]

A pointer to a function cannot be used in an addition/subtraction nor comparison operation. Continues the compilation regarding it as a pointer to an object.

W1007C	empty declaration
--------	-------------------

[Explanation]

A semicolon alone is written outside the function. A declaration is to be written outside the function. Continues the compilation neglecting the semicolon.

W1008C	unknown size of incomplete type
--------	---------------------------------

[Explanation]

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.

W1009C	unknown size of incomplete type
--------	---------------------------------

[Explanation]

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.

W1010C	unknown size of function type
--------	-------------------------------

[Explanation]

Attempted to get the size of the function type. It's impossible, so 1 is assumed.

Remark: This message is sub message of W1124C,W1125C.

W1011C	unknown size
--------	--------------

[Explanation]

Attempted to get the size of the void type. It's impossible, so 1 is assumed.

Remark: This message is sub message of W1124C,W1125C.

W1012C	invalid null subscript of array type
--------	--------------------------------------

[Explanation]

Attempted to get the size of the array type whose length is unknown. Zero is assumed.

Remark: 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.

Remark: 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.

Remark: 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.

W1016C	%Z: array dimension is variable
--------	---------------------------------

[Explanation]

Only an integral constant can be specified for a subscript of an array in a declaration.

Remark: This message is not used.

W1017C	%Z: brace-enclosed list of initializers expected for aggregate type
--------	---

[Explanation]

An initial value for a struct, union or array is not enclosed with "{}". Continue the compilation making it up.

W1018C	%Z: too long %s literal (%u) for array: `%D'
--------	--

[Explanation]

Too long string literal is specified for the initial value to initialize an array with a character string or an 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.

W1019C	%Z: address of string constant is used for `%D'
--------	---

[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.

W1020C	%Z: address of static symbol `%D' is used for symbol `%D'
--------	---

[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.

W1021C	parameter '%D' unused
--------	-----------------------

[Explanation]

A parameter is not referred in the function. Continues the compilation making the parameter valid.

W1022C	'%D' unused but it is set
--------	---------------------------

[Explanation]

A value is set to a local variable but not referred. Continues the compilation making the variable valid.

W1023C	'%D' unused
--------	-------------

[Explanation]

A local variable is not referred. Continues the compilation making the variable valid.

W1024C	inefficient operation: result value may be ignored: %O
--------	--

[Explanation]

An operation which has no side effect is specified where a side effect is expected. Continues the compilation leaving it as it is.

W1025C	statement not reached
--------	-----------------------

[Explanation]

A statement is never reached because of a branch etc. Continues the compilation leaving it as it is.

W1026C	non void type function '%D' is expected to return value
--------	---

[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.

W1027C	too large long long integer constant %X
--------	---

[Explanation]

An integral constant exceeds the maximum value that this system supports. Continues the compilation assuming that the maximum value is specified.

Remark: This message is not used.

W1028C	empty translation unit
--------	------------------------

[Explanation]

A compile unit must contain at least one external declaration. Continues the compilation leaving it as it is.

W1029C	enumerator list has optional comma
--------	------------------------------------

[Explanation]

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.

W1030C	unknown # directive
--------	---------------------

[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.

The indicated directive is unknown for the compiler. Continues the compilation neglecting the directive.

W1031C	too long identifier, truncated to `%s'
--------	--

[Explanation]

The length of the identifier is larger than the maximum value(65534) the system can manage. Continues the compilation neglecting the excess part.

W1032C	invalid digit postfix expression
--------	----------------------------------

[Explanation]

An invalid postfix is specified in the integral constant. Continues the compilation neglecting the postfix.

Remark: This message is not used.

W1033C	integer constant out of range
--------	-------------------------------

[Explanation]

The integral constant is out of range for the type specified by the postfix. Continues the compilation leaving it as it is.

W1034C	`;' expected at the end of member declaration
--------	---

[Explanation]

The `;' is missing after the last member declaration of a struct or union tag declaration. Though the last `;' can be omitted in some old grammar, not in the ANSI. Continues the compilation assuming that the `;' is specified.

W1035C	character constant too long
--------	-----------------------------

[Explanation]

The wide character constant is too long. Continues the compilation making the first character valid.

W1036C	newline in string literal
--------	---------------------------

[Explanation]

A double quotation to terminate the string literal is not specified in the line. Search for it in lines below.

W1037C	numeric octal constant contains `8' or `9'
--------	--

[Explanation]

The `8' or `9' is specified in an octal integer constant. Continues the compilation regarding the `8' as `010' and the `9' as `011'.

W1038C	invalid character `\\%o' (octal)
--------	----------------------------------

[Explanation]

An invalid character is found in a source program. Continues the compilation neglecting the character.

W1039C	modulus by 0
--------	--------------

[Explanation]

A division by zero is performed in a modulus operation. The object for the operation is output. Modify the source code.

W1040C	division by 0
--------	---------------

[Explanation]

A division by zero is performed. The object for the operation is output. Modify the source code.

W1041C	type mismatch in function prototype
--------	-------------------------------------

[Explanation]

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.

W1042C	ellipsis mismatch in function parameter: parameter %d
--------	---

[Explanation]

Positions the ellipsis is specified do not correspond between function prototype declarations. Continues the compilation neglecting the declaration.

W1043C	type mismatch in function prototype: parameter %d
--------	---

[Explanation]

Types of parameters do not correspond between function prototype declarations. Continues the compilation neglecting the declaration.

W1044C	number of function parameters is different: parameter %d
--------	--

[Explanation]

Numbers of parameters do not correspond between function prototype declarations. Continues the compilation neglecting the declaration.

W1045C	incompatible types: assumed that plain `char' and `%T' are compatible types
--------	---

[Explanation]

The type is incomplete according to the ANSI.

Remark: This message is not used.

W1046C	invalid character `%o' (octal) in # directive line
--------	--

[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.

Remark: This message is not used.

W1047C	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.

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.

Remark: 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 an 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.

Remark: This message is not used.

W1053C	incompatible types: assumed that pointer to char and pointer to void are compatible types
--------	---

[Explanation]

Types of a pointer to char and a pointer to void are regarded to be compatible. Continues the compilation assuming that they are compatible types.

Remark: This message is not used.

W1054C	incompatible types between pointer to void and pointer to function
--------	--

[Explanation]

According to the ANSI types of a pointer to void and a pointer to function are regarded to be incompatible. Continues the compilation assuming that they are compatible types.

W1055C	incompatible types: assumed that pointer to void and pointer to non-void are compatible types
--------	---

[Explanation]

Types of a pointer to void and a pointer to non-void are regarded to be compatible. Continues the compilation assuming that they are compatible types.

Remark: This message is not used.

W1056C	invalid field '%T' is used
--------	----------------------------

[Explanation]

Though the indicated member name is not found in any struct nor union, it is found as a plain symbol. Continues the compilation assuming that it is a plain symbol according to the old specification.

W1057C	cast to incomplete type '%T'
--------	------------------------------

[Explanation]

An enumeration type whose tag is not defined is specified as the target type of a type conversion. Continues the compilation assuming that it is converted to the int type.

W1058C	identifier '%D' is used out of its scope
--------	--

[Explanation]

An identifier which is declared as an external reference symbol in a block is referred outside the block. Continues the compilation making the reference valid.

W1059C	This behavior is ANSI C undefined behavior
--------	--

[Explanation]

The operation is regarded as undefined in the ANSI C.

Remark: This message is not used.

W1060C	length of external linkage identifier (%D %d) is longer than %d
--------	---

[Explanation]

The length of an identifier for the external linkage is too long.

Remark: This message is not used.

W1061C	length of internal linkage identifier (%D %d) is longer than %d
--------	---

[Explanation]

The length of an identifier for the internal linkage is too long.

W1062C	void type parameter name '%D' is ignored
--------	--

[Explanation]

If the type of a parameter for a function is void, no identifier is needed for a parameter. Continue the compilation neglecting the identifier.

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.

W1068C	case value is out of range: type of case expression is inconsistent with type of switch conditional expression: case value is truncated
--------	---

[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.

W1069C	decimal integer constant '%s' is too large
--------	--

[Explanation]

The decimal integer constant is too large to express internally. Continue the compilation making lower bits valid.

W1070C	octal integer constant '%s' is too large
--------	--

[Explanation]

The octal integer constant is too large to express internally. Continues the compilation making lower bits valid.

W1071C	hexadecimal integer constant '0x%s' is too large
--------	--

[Explanation]

The hexadecimal integer constant is too large to express internally. Continues the compilation making lower bits valid.

W1072C	binary integer constant '%s' is too large
--------	---

[Explanation]

The binary integer constant is too large to express internally. Continues the compilation making lower bits valid.

Remark: This message is not used.

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.

Remark: 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.

Remark: 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.

W1078C	'%s' is ANSI C extension
--------	--------------------------

[Explanation]

The "__asm" is specification of the ANSI C extension. Continues the compilation leaving it as it is.

Remark: This message is not used.

W1079C	prototype parameter declaration is used
--------	---

[Explanation]

A function prototype is used. Continues the compilation leaving it as it is.

Remark: This message is not used.

W1080C	`%s' is used
--------	--------------

[Explanation]

An asm statement is found. Continues the compilation leaving it as it is.

W1081C	multiple type qualifier specified in declaration with obsolete modified typedef type
--------	--

[Explanation]

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.

Remark: This message is not used.

W1082C	multiple `const' specified
--------	----------------------------

[Explanation]

Several "const", which effects on a symbol or pointer, are specified. The effect of all of them is equal to that of one "const".

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.

W1089C	lacked tag name in struct/union declaration
--------	---

[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.

W1090C	tag is declared first in parameter list: `%'E %'I' has function prototype scope
--------	---

[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.

W1091C	redeclaration of `D`
--------	----------------------

[Explanation]

A symbol is redeclared. Assure that the type is compatible.

W1092C	redeclaration of `D`: linkage conflict
--------	--

[Explanation]

The declaration of the symbol is regarded as a redeclaration because the linkage conflicts. Check a storage class specifier.

W1093C	redeclaration of builtin symbol `D` as static function
--------	--

[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.

W1094C	redeclaration of `D`: promoted parameter mismatch
--------	---

[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.

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.
Remark: 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.

W1101C	local declaration of `'%D'` hides parameter
--------	---

[Explanation]

The symbol which is already declared as a parameter is declared as a local declaration. It is a redeclaration according the standard.

W1102C	declaration of `'%D'` hides previous declaration
--------	--

[Explanation]

The declaration of a symbol in a block makes the same symbol declared outside the block invisible. The symbol declared outside is invisible.

W1103C	subscript of array is zero
--------	----------------------------

[Explanation]

The value of a subscript of an array is equal to zero. Continues the compilation making the declaration valid.

W1104C	useless type qualifiers for function return type: `"%T"
--------	---

[Explanation]

A type qualifier (const, volatile) specified to a return value of a function is meaningless.

Remark: This message is not used.

W1105C	useless type qualifier specified
--------	----------------------------------

[Explanation]

A type qualifier is specified to a tag declaration. A type qualifier is effective for a symbol or a pointer and meaningless here.

W1106C	function return block scope type: `"%T"
--------	---

[Explanation]

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.

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.

Remark: 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.

Remark: This message is not used.

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.

Remark: 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.

W1123C	%Z incompatible pointer type in function `%D'
--------	---

[Explanation]

Types of pointers are incompatible.
Remark: This message is not used.

W1124C	%Z: %s: `%T': %O
--------	------------------

[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.

W1125C	%Z: %s: `%T'
--------	--------------

[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.

W1126C	%Z incompatible pointer type: `%D'
--------	------------------------------------

[Explanation]

Types of pointers are incompatible.
Remark: This message is not used.

W1127C	type conversion void to void
--------	------------------------------

[Explanation]

A cast operation is made from the void type to the void type.

Remark: This message is not used.

W1128C	invalid type conversion to void type
--------	--------------------------------------

[Explanation]

A cast operation is made to the void type.

Remark: This message is not used.

W1129C	invalid type conversion to same non scalar type
--------	---

[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.

W1130C	invalid type for conditional expression: '%T' specified: scalar type expected: %O
--------	---

[Explanation]

The void type is specified to the first operand of the conditional operator.

Remark: This message is not used.

W1131C	'%D' may be used before it is set
--------	-----------------------------------

[Explanation]

A symbol is referred before its value is set. The resulting value is not safe.

W1132C	implicitly function declared '%T': assumed return 'int'
--------	---

[Explanation]

A function is called before declared. The type of the return value is assumed as the 'int' type.

W1133C	return without value in non void type function '%D'
--------	---

[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.

Remark: 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.

Remark: 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.

W1145C	%O applies 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.

Remark: 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.

Remark: This message is not used.

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.
Remark: This message is not used.

W1156C	%Z between incompatible pointer types: %O
--------	---

[Explanation]

An incompatible pointer.
Remark: This message is not used.

W1157C	non void type function `%D' reaches to the end of function without return
--------	---

[Explanation]

No return statement is written in a function of a non-void type. An unknown value is returned.

W1158C	function '%D' returns 'int'
--------	-----------------------------

[Explanation]

No declaration specifier is written in a function definition. Assume that the function returns an int type value, for no type information is available.

W1159C	linkage conflict between internal and external
--------	--

[Explanation]

The linkage does not match with that defined symbol previously.

Remark: This message is sub message of W1096C,W1097C,W1098C.

W1160C	%Z: left hand side must be modifiable lvalue
--------	--

[Explanation]

The left-hand side must be modifiable lvalue.

W1161C	reference to rvalue array
--------	---------------------------

[Explanation]

An array as rvalue is referred.

W1162C	storage class specifier '%A' is specified for function '%D' declared in block
--------	---

[Explanation]

A function is declared with the "static" storage class specifier in a block. According to the standard, an explicit storage class specifier except for the "extern" cannot be specified in a function declaration in a block.

W1163C	extern function '%D' declared in block
--------	--

[Explanation]

A function is declared with the "extern" storage class specifier in a block.

W1164C	`long long' integer constant is used
--------	--------------------------------------

[Explanation]

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.

W1165C	parameter type mismatch of `%D'
--------	---------------------------------

[Explanation]

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.

W1166C	value of integral constant expression for enumerator `%I' is out of range
--------	---

[Explanation]

The number set to an enumerator constant exceeds the INT_MAX. The number must be expressible by the `int' type.

W1167C	unary minus operator applys to too large unsigned integer constant
--------	--

[Explanation]

The unary '-' operator is applied to an integral constant of the unsigned type.

Remark: This message is not used.

W1168C	`%E %I' declared in parameter declaration
--------	---

[Explanation]

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.

W1169C	label `%D' is not referred
--------	----------------------------

[Explanation]

The label is not referred by any goto statement. It does not bring a problem to a program.

W1170C	`long long' type specifier is used
--------	------------------------------------

[Explanation]

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.

W1171C	unterminated filename
--------	-----------------------

[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. 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.

W1172C	too long hexadecimal escape sequence
--------	--------------------------------------

[Explanation]

The hexadecimal by the escape sequence ``x'` cannot be expressed by a character. Continues the compilation making only the lowest byte or wide-character of the resulting value valid.

W1173C	<code>`*/'</code> exists outside of comment
--------	---

[Explanation]

Though a comment must be enclosed by the `"/*` and `*/"` in the C language, `*/"` is found outside the comment. A comment cannot be nested. Continues the compilation regarding those simply as the ``*' and the `/'.`

W1174C	hexadecimal escape sequence has no digit value
--------	--

[Explanation]

No hexadecimal digit is written after the ``x'` escape sequence. Continues the compilation assuming that a ``x0'` is written.

W1175C	type conversion between different integral types
--------	--

[Explanation]

A type conversion is made between integral types with different sizes.

Remark: This message is not used.

W1176C	type conversion between different floating-points
--------	---

[Explanation]

A type conversion is made between floating point types with different sizes.

W1177C	truncate pointer to lose significance: `%s%s'
--------	---

[Explanation]

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.

W1178C	integral type expression expected for `%%%c': function `%s' argument %d
--------	---

[Explanation]

A pointer (character string) to the `char' type cannot be specified for conversion specifiers as %d, %u, &o and %i in a format character string of the printf.

W1179C	integral type expression expected for `%%%c': function `%s' argument %d
--------	---

[Explanation]

A pointer cannot be specified for the conversion specifier %c in a format character string of the printf.

W1180C	pointer to void type expression expected for `%%%c': function `%s' argument %d
--------	--

[Explanation]

Only the pointer to the void type can be specified for the conversion specifier %p in a format character string of the printf.

W1181C	pointer to integral type expression expected for `%%%c': function `%s' argument %d
--------	--

[Explanation]

Only the pointer to the integral type can be specified for the conversion specifier %n in a format character string of the printf.

W1182C	too many arguments for format string: function '%s'
--------	---

[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]

The digit-sequence is too long and surely overflows before converting it into the internal format. Continues the compilation assuming that 0 is specified.

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.

Remark: 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.

Remark: This message is not used.

W1193C	different symbol types between '%s' and '%s'
--------	--

[Explanation]

The same identifier is used in different ways in the same name space.

W1194C	unknown format flags 0x%x: function '%s'
--------	--

[Explanation]

An unknown conversion specifier is found in a format character string of the printf.

W1195C	unknown format flags %%%c: function '%s'
--------	--

[Explanation]

An unknown conversion specifier is found in a format character string of the printf.

W1196C	function return type '%T' is not promoted for function type declaration without prototype
--------	---

[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.

W1197C	integer constant isn't treated as unsigned: %O
--------	--

[Explanation]

Regard the unsigned integral constant as a signed one.

W1198C	'%T' is useless declaration
--------	-----------------------------

[Explanation]

It is meaningless to declare an enumeration tag (enum tag) alone. The tag is ignored.

W1199C	array of incomplete type '%T'
--------	-------------------------------

[Explanation]

Though a struct or union type is specified as an element of an array, it is an incomplete type (the tag is not defined).

W1200C	suspicious assignment operator in conditional expression: %O
--------	--

[Explanation]

An assignment operator is written in a condition expression. Assure that it is not intended as the '==' operator.

W1201C	different semantics between transition and later: %O
--------	--

[Explanation]

The effect of the operation is changed owing to the transition from the unsigned preserving rule to the value preserving rule.

W1202C	call function without prototype declaration: '%s'
--------	---

[Explanation]

A function which is not prototyped is called. Assure that arguments are specified correctly.

W1203C	use address of string constant in conditional expression
--------	--

[Explanation]

An address of a character string is specified as an operand of a logical operation.

W1204C	static symbol '%D' unused
--------	---------------------------

[Explanation]

A symbol declared with the "static" storage class specifier is not referred. A region for the symbol is reserved because it is a static symbol.

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.

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.

Remark: 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.

Remark: 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.

W1229C	floating point exception: %O
--------	------------------------------

[Explanation]

An arithmetic operation results in a floating point exception. Inspect a constant operation.

W1230C	floating point division by 0: %O
--------	----------------------------------

[Explanation]

A floating point division by zero is performed in an arithmetic operation. Inspect a constant operation.

W1231C	floating point overflow: %O
--------	-----------------------------

[Explanation]

In an arithmetic operation, a floating point overflow occurred. Inspect a constant operation.

W1232C	floating point underflow: %O
--------	------------------------------

[Explanation]

In an arithmetic operation, a floating point underflow occurred. Inspect a constant operation.

W1233C	floating point inexact: %O
--------	----------------------------

[Explanation]

In an arithmetic operation, the result of a floating point operation is not exact. Inspect a constant operation.

W1234C	conversion to `%T' is out of range
--------	------------------------------------

[Explanation]

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.

W1235C	arithmetic calculation is out of range: %O
--------	--

[Explanation]

In an arithmetic operation, the resulting value of an operation is out of range. Inspect a constant operation.

W1236C	invalid type combination of `%T' and `%T': %O
--------	---

[Explanation]

The combination for types of operands is invalid.

W1237C	invalid type combination of '%T' and '%T': integral type required: %O
--------	---

Explanation]

The combination for types of operands is invalid.

W1238C	invalid type combination of '%T' and '%T': arithmetic type required: %O
--------	---

Explanation]

The combination for types of operands is invalid.

W1239C	invalid type combination of '%T' and '%T': void type required: %O
--------	---

Explanation]

The combination for types of operands is invalid.

W1240C	invalid type combination of '%T' and '%T': compatible types required: %O
--------	--

Explanation]

The combination for types of operands is invalid.

W1241C	unknown escape sequence '\%o'
--------	-------------------------------

Explanation]

A character which is not specified as an escape sequence in the ANSI C is found after the `\`. Continues the compilation neglecting the `\`.

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".

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.

W1253C	__io do not operate on external variable
--------	--

[Explanation]

The "__io" is effective only on an external reference variable.

Remark: This message is not used.

W1254C	__io operate on function declarator
--------	-------------------------------------

[Explanation]

The "__io" cannot be specified to a function declarator. Assure that a valid type qualifier is specified. Continues the compilation neglecting the "__io".

W1255C	__io operate struct or union member
--------	-------------------------------------

[Explanation]

The "__io" is effective only on a variable. A part of members for a struct or union cannot be qualified by the "__io". The "__io" is ignored.

W1256C	__interrupt is specified
--------	--------------------------

[Explanation]

The "__interrupt" is the extended specification.

W1257C	__subinterrupt is specified
--------	-----------------------------

[Explanation]

The "__subinterrupt" is the extended specification.

W1258C	__io is specified
--------	-------------------

[Explanation]

The "__io" is the extended specification.

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.

Remark: 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.

W1265C	pointer type required for argument: %D
--------	--

[Explanation]

An argument which specifies a format character string is not the pointer type. The argument is not checked.

W1266C	no struct members have name
--------	-----------------------------

[Explanation]

None of members for a struct has a name. At least one must have a name.

W1267C	no union members have name
--------	----------------------------

[Explanation]

None of members for a union has a name. At least one must have a name.

W1268C	shift count %X is negative: %O
--------	--------------------------------

[Explanation]

The integral constant specified as a shift value of a shift operation is negative. The shift operation is performed badly.

W1269C	shift count %X is too large for `%%T': %O
--------	---

[Explanation]

The integral constant specified as a shift value of a shift operation is larger than the size of the type for an operand.

W1270C	size of member `%%D' is zero
--------	------------------------------

[Explanation]

The size of a member must be positive.

W1271C	`%%T' is used
--------	---------------

[Explanation]

The "int", "signed int" or "unsigned int" type is specified. The size of the int type differs for systems, then it is recommended to specify "short" or "long" simultaneously.

W1272C	`%%T' is used for declaration `%%D'
--------	-------------------------------------

[Explanation]

The "int" or "unsigned int" type is specified. The size of the int type differs for systems, then it is recommended to specify "short" or "long" simultaneously.

W1273C	comparison between pointer and constant: %O
--------	---

[Explanation]

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.

W1274C	comparison between NULL and address of data object or function: %O
--------	--

[Explanation]

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.

W1275C	assignment from '%T' to '%T'
--------	------------------------------

[Explanation]

An assignment operation to the "int" or "unsigned int" type is performed. The size of the int type differs for systems, then it is recommended to specify "short" or "long" simultaneously.

W1276C	cast expression cannot be lvalue
--------	----------------------------------

[Explanation]

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.

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.
Remark: This message is not used.

W1280C	'long double' used: %O
--------	------------------------

[Explanation]

The long double type is specified.
Remark: This message is not used.

W1281C	comparison between pointers: %O
--------	---------------------------------

[Explanation]

A comparison between pointers.

Remark: This message is not used.

W1282C	misalign declaration of `double`/`long double` struct/union member: `%D'
--------	--

[Explanation]

The boundary alignment of a member for the double or long double type is invalid.

Remark: This message is not used.

W1283C	layout of bit-field depends on application binary interface: `%D'
--------	---

[Explanation]

The layout of a bit-field depends on the application binary interface.

Remark: This message is not used.

W1284C	bit-field declaration without signed and unsigned: `%D'
--------	---

[Explanation]

A bit-field is declared without signed or unsigned specified.

Remark: This message is not used.

W1285C	bit-field declaration with enumerate type: `%D'
--------	---

[Explanation]

A bit-field of the enumeration type is declared.

Remark: This message is not used.

W1286C	useless type qualifier of `%T'
--------	--------------------------------

[Explanation]

A type qualifier specified with the "void" type is meaningless. Ignore the type qualifier.

W1287C	invalid void type parameter declaration: `%D'
--------	---

[Explanation]

A void type is invalid in a parameter declaration.

Remark: This message is not used.

W1288C	too large object `%D'
--------	-----------------------

[Explanation]

The size of the object is too large.

Remark: This message is not used.

W1289C	`...' in expression violates ANSI C specification
--------	---

[Explanation]

The `...' can be specified only in a parameter list of a function declaration. Modify the expression.

W1290C	too long multi-character character constant
--------	---

[Explanation]

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.

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.

Remark: 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.

Remark: 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.

Remark: This message is not used.

W1299C	#pragma: invalid switch 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, the specified parameter for the pragma is meaningless. Continues the compilation neglecting the "#pragma" directive.

Remark: 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.

Remark: 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.

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.

Remark: This message is not used.

W1308C	#error directive used
--------	-----------------------

[Explanation]

The "#error" directive is specified.

W1309C	#pragma locale: 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 locale" is one of those directives, this pragma needs a string literal for a parameter. Continues the compilation neglecting the "#pragma" directive.

W1310C	#sccs directive used
--------	----------------------

[Explanation]

The "#sccs" directive is specified.

W1311C	#ident directive used
--------	-----------------------

[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 is an old style. It is recommended to replace it by "#pragma ident". Continues the compilation leaving it as it is.

W1312C	#pragma: unknown pragma
--------	-------------------------

[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, no pragma name is specified. Continues the compilation neglecting the "#pragma" directive.

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.

Remark: This message is not used.

W1317C	#pragma: unknown specifier
--------	----------------------------

[Explanation]

An invalid parameter is specified in the "#pragma" directive.

Remark: This message is not used.

W1318C	#pragma %s: unknown optimization specifier
--------	--

[Explanation]

An invalid parameter is specified in the "#pragma" directive.

Remark: This message is not used.

W1319C	#pragma %s: invalid argument
--------	------------------------------

[Explanation]

An invalid parameter is specified in the "#pragma" directive.

Remark: This message is not used.

W1320C	#pragma %s: identifier expected
--------	---------------------------------

[Explanation]

An identifier is needed for a parameter of the "#pragma" directive.

Remark: This message is not used.

W1321C	#pragma %s: invalid token: identifier expected
--------	--

[Explanation]

An identifier is needed for a parameter of the "#pragma" directive.

Remark: This message is not used.

W1322C	#pragma %s: invalid token: too many parameters
--------	--

[Explanation]

Too many parameters are specified in the "#pragma" directive.

W1323C	#pragma %s: integer constant expected
--------	---------------------------------------

[Explanation]

An integral constant is needed for a parameter of the "#pragma" directive.

Remark: This message is not used.

W1324C	#pragma %s: invalid token: integer constant expected
--------	--

[Explanation]

An integral constant is needed for a parameter of the "#pragma" directive.

Remark: This message is not used.

W1325C	#pragma %s: invalid optimization specifier
--------	--

[Explanation]

An invalid optimization specifier is specified in the "#pragma" directive.

Remark: This message is not used.

W1326C	#pragma section: syntax error: `=' is expected
--------	--

[Explanation]

The `=' is needed just after the "attr" parameter or the "locate" parameter of the "#pragma section" directive. Continues the compilation neglecting the "#pragma section".

W1327C	#pragma section: invalid hexadecimal constant
--------	---

[Explanation]

An invalid value is set for the locate parameter of the "#pragma section".

Remark: This message is not used.

W1328C	#pragma section: invalid sectionname specified
--------	--

[Explanation]

An unknown identifier is specified for a section name of the "#pragma section". Continues the compilation neglecting the "#pragma section".

W1329C	#pragma section: invalid sectionattr specified
--------	--

[Explanation]

An invalid identifier is specified for the attr parameter of the "#pragma section". Continues the compilation neglecting the "#pragma section".

W1330C	#pragma asm: syntax error: unknown specifier
--------	--

[Explanation]

An excess character is found after the "#pragma asm".

Remark: This message is not used.

W1331C	#pragma section: too long identifier is specified
--------	---

[Explanation]

Too long identifier is specified for a section name of the "#pragma section" . Continues the compilation making the first 255 bytes valid.

W1332C	#pragma asm: syntax error: `#pragma endasm' is not specified
--------	--

[Explanation]

The "#pragma endasm" is missing.

Remark: This message is not used.

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".

Remark: 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".

Remark: 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.

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.

W1351C	%Z incompatible pointer types: expected `%T' actual `%T': argument %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.

W1352C	%Z incompatible pointer types from `%T' to `%T': `%D'
--------	---

[Explanation]

Types of an initializer and of a target are incompatible. Modify the initializer to match with the target type.

W1353C	%Z incompatible pointer types from `%T' to `%T': %O
--------	---

[Explanation]

Types of both sides of the assignment operation are incompatible. Modify the right-hand side value to match with the type of the left-hand side value.

W1354C	%Z: constant out of range for `%T' bit-field: its width %d
--------	--

[Explanation]

A constant specified as an initializer exceeds the range of a bit-field. Modify the constant value.

W1355C	multiple `__far' specified
--------	----------------------------

[Explanation]

Several "__far", which is effective on a symbol or a pointer, are specified. Only one is effective.

W1356C	multiple `__near' specified
--------	-----------------------------

[Explanation]

Several "__near", which is effective on a symbol or a pointer, are specified. Only one is effective.

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".

W1363C	__near operate struct or union member
--------	---------------------------------------

[Explanation]

The "__near" 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 "__near".

W1364C	__far is specified
--------	--------------------

[Explanation]

The "__far" is the extended specification.

W1365C	__near is specified
--------	---------------------

[Explanation]

The "__near" is the extended specification.

W1366C	__direct is specified
--------	-----------------------

[Explanation]

The "__direct" is the extended specification.

W1367C	__direct operate on function declarator
--------	---

[Explanation]

The "__direct" is not applicable to a function declarator. Check the specification of the type qualifier. Continues the compilation neglecting the "__direct".

W1368C	__direct operate struct or union member
--------	---

[Explanation]

The "__direct" is effective only on a variable. It is not applicable to a member of a struct or union. Continues the compilation neglecting the "__direct".

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.
Remark: This message is not used.

W1373C	indirect reference function call specified
--------	--

[Explanation]

A function is called through a function pointer.

W1374C	#pragma %s: already exist
--------	---------------------------

[Explanation]

The indicated “#pragma” is already specified. It cannot be nested. Continues the compilation neglecting the indicated pragma.

W1375C	`#pragma %s' expected
--------	-----------------------

[Explanation]

The indicated “#pragma” is not specified. Continues the compilation assuming that the pragma is specified at the end of the compile unit.

W1376C	#pragma intvect: invalid constant for vector number
--------	---

[Explanation]

An invalid value is specified for a vector number of the "#pragma intvect". Continues the compilation neglecting the "#pragma intvect".

W1377C	#pragma intvect: invalid constant for mode number
--------	---

[Explanation]

An invalid value is specified for a mode of the "#pragma intvect". Continues the compilation neglecting the mode specification.

W1378C	#pragma intvect: mode number is out of range
--------	--

[Explanation]

The value of a mode specified in the "#pragma intvect" is larger than 255. Continues the compilation neglecting the mode specification.

W1379C	#pragma intvect: difference mode number for existed same vector number
--------	--

[Explanation]

Though the "#pragma intvect" is already specified for the same function name with the same vector number, a mode is specified differently. Continues the compilation neglecting the "#pragma intvect".

W1380C	#pragma section: invalid constant for allocated address
--------	---

[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]

An initial value cannot be set in the area where a variable which is qualified by the "__io" is allocated. Continues the compilation making the initialization invalid.

W1386C	different size: argument type `%T' with prototype `%T': argument %d
--------	---

[Explanation]

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.

W1387C	different unsignedness: argument type `%T' with prototype `%T': argument %d
--------	---

[Explanation]

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.

W1388C	Ignore since the 255th character of symbol `%s'
--------	---

[Explanation]

The external symbol name is too long. The first 254 characters are valid.

W1389C	#pragma intvect: vector number is out of range
--------	--

[Explanation]

The vector number specified in the "#pragma intvect" exceeds the maximum value in this system. Continues the compilation assuming that no value is specified.

W1390C	restrict is specified
--------	-----------------------

[Explanation]

The "restrict" is the extended specification.

W1391C	multiple `restrict' specified
--------	-------------------------------

[Explanation]

The "restrict" 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 "restrict".

W1392C	typedefed type already qualified with `restrict'
--------	--

[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.

W1393C	%Z: exist data without initial value: `%D'
--------	--

[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.

W1394C	#pragma %s: specify 1 or 2 for align value
--------	--

[Explanation]

#pragma %s: specify 1 or 2 for align value.

E4001C	type of function designator must be function type
--------	---

[Explanation]

A type other than the function type nor the function pointer type is specified in the function call.

E4002C	pointer or array type are expected
--------	------------------------------------

[Explanation]

A type other than the pointer type nor the array type is specified as pointer reference (* operator).

E4003C	pointer to struct/union type required for left operand
--------	--

[Explanation]

Specify a pointer to the struct or union.

Remark: This message is not used.

E4004C	struct/union type required for left operand
--------	---

[Explanation]

Specify struct or union type.

Remark: 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.

E4009C	unknown size of incomplete type
--------	---------------------------------

[Explanation]

The struct, union or enumeration tag has no member, and no size information is available.

E4010C	unknown size of array type
--------	----------------------------

[Explanation]

A type with an unknown size is specified as an element of an array. Cannot determine the total size of the array.

E4011C	unknown size of function type
--------	-------------------------------

[Explanation]

Attempted to get the size of the function type. It's impossible, so 1 is assumed.

Remark: This message is not used.

E4012C	unknown size
--------	--------------

[Explanation]

Attempted to get the size of the void type. It's impossible, so 1 is assumed.

Remark: This message is not used.

E4013C	invalid null subscript of array type
--------	--------------------------------------

[Explanation]

Attempted to get the size of the array type with unknown size. 0 is assumed.

Remark: This message is not used.

E4014C	invalid variable subscript of array
--------	-------------------------------------

[Explanation]

The subscript of the array is not an integral constant. Cannot determine the size of the array.

E4015C	invalid label
--------	---------------

[Explanation]

An invalid label is specified in the goto statement.

E4016C	undefined label '%D' is used as destination
--------	---

[Explanation]

The label specified in the goto statement does not exist in the function.

E4017C	label '%D' redefined
--------	----------------------

[Explanation]

The label name is already defined in the function. A label name must be unique in a function scope.

E4018C	%Z: too many initializers for '%T': '%D'
--------	--

[Explanation]

The number of initializers of the struct or union is larger than that of members. Especially, only one initializer for the first member may be specified in the union.

E4019C	%Z: incomplete type '%T' specified: '%D'
--------	--

[Explanation]

Attempted to initialize a struct or union whose tag is not declared.

E4020C	%Z: invalid array null dimension: '%D'
--------	--

[Explanation]

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.

E4021C	%Z: invalid array variable dimension: '%D'
--------	--

[Explanation]

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.

E4022C	%Z: invalid initializer: `%D'
--------	-------------------------------

[Explanation]

An object of the function type cannot be specified as an initializer.

E4023C	%Z: invalid initializer for array: `%D'
--------	---

[Explanation]

Initializer is badly specified for an array.

E4024C	%Z: invalid initializer for function `%D'
--------	---

[Explanation]

Initializer is specified for the function declaration. A function cannot be initialized.

E4025C	%Z: invalid initializer for struct/union: `%D'
--------	--

[Explanation]

The initializer for the struct or union is specified in a bad manner.

E4026C	%Z: invalid initializer for `%T': `%D'
--------	--

[Explanation]

Only a symbol must be specified to initialize a struct or union with another struct or union of a compatible type.

E4027C	%Z: invalid initializer for struct/union
--------	--

[Explanation]

The type of the struct or union specified as an initializer is incompatible with that of the target struct or union.

E4028C	%Z: too many initializers for array: `%D'
--------	---

[Explanation]

More initializers are specified than the subscript value of array.

E4029C	%Z: too deeply nested brace: `%D'
--------	-----------------------------------

[Explanation]

Too many braces are specified to enclose the initializer. Modify to fit for the type of the target.

E4030C	%Z: integral constant expression required for bit-field `%D': `%D'
--------	--

[Explanation]

Only an integral constant can be specified as an initializer of a bit-field.

E4031C	%Z: too many initializers for an element: `%D'
--------	--

[Explanation]

Too many initializers are specified. Only one initializer is needed for one variable or member.

E4032C	%Z: constant expression is expected for %s: `%D'
--------	--

[Explanation]

Only a constant expression can be specified as an initializer.

E4033C	`=' expected before initializer
--------	---------------------------------

[Explanation]

The compound assignment operator is specified prior to assigning an initial value. Only a simple assignment operator can be used in the initialization.

E4034C	`=' expected before enumerator initializer
--------	--

[Explanation]

The compound assignment operator is specified prior to initializing an enumerator. Only a simple assignment operator can be used in an initialization.

E4035C	`continue' found outside loops
--------	--------------------------------

[Explanation]

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.

E4036C	`break' found outside switches and loops
--------	--

[Explanation]

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.

E4037C	unterminated string literal: `\" expected
--------	---

[Explanation]

The string literal does not end.
Remark: This message is not used.

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.

E4044C	too many postfix characters '%c' for constant
--------	---

[Explanation]

The postfix of the numerical constant is not specified correctly.

E4045C	numeric octal constant contains invalid character
--------	---

[Explanation]

A character which is not an octal digit is used in the octal constant.

Remark: This message is not used.

E4046C	binary constant cannot be floating point constant
--------	---

[Explanation]

Binary constant beginning with '0b' or '0B' cannot be floating point constant.

Remark: This message is not used.

E4047C	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.

E4048C	invalid postfix character `'%c' for radix %d
--------	--

[Explanation]

A character which is not an alphabet and cannot be recognized as a postfix is specified at the end of the integral constant.

E4049C	no digits of floating exponent part
--------	-------------------------------------

[Explanation]

No digit is specified for the exponent part of the floating point number.

E4050C	hexadecimal constant cannot be floating point constant
--------	--

[Explanation]

The `.' is found in the hexadecimal constant.

E4051C	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.

E4052C	invalid token: `..'
--------	---------------------

[Explanation]

An invalid token `..' is found. The `.' is an operator, the `...' an ellipsis.

E4053C	integer constant out of range
--------	-------------------------------

[Explanation]

The integral constant cannot be expressed internally.

Remark: This message is not used.

E4054C	invalid character `'%c'
--------	-------------------------

[Explanation]

An invalid character is found. It is not included in the source character set.

E4055C	invalid binary constant
--------	-------------------------

[Explanation]

The first digit of the binary constant beginning with `0b' or `0B' is not 0 nor 1.

Remark: 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.

E4061C	%s near wide character constant
--------	---------------------------------

[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.

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.

Remark: This message is not used.

E4069C	incompatible subscript between variables
--------	--

[Explanation]

Subscripts of array are declared differently. Regard that types are incompatible.

Remark: 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.

E4072C	different data types specified for declaration of '%s'
--------	--

[Explanation]

Different data type specifiers are specified at once. Only one can be specified.

E4073C	invalid short or long specified with char for declaration of '%s'
--------	---

[Explanation]

The type "char" cannot be specified with "short" nor "long".

E4074C	invalid short or long specified for declaration of `%s'
--------	---

[Explanation]

The type specifier cannot be specified with "short" nor "long".

E4075C	invalid signed or unsigned specified for declaration of `%s'
--------	--

[Explanation]

The type specifier cannot be specified with "signed" nor "unsigned".

E4076C	multiple storage class specifier specified for declaration of `%s'
--------	--

[Explanation]

Several storage class specifiers are written. Only one can be specified.

E4077C	invalid long specified for declaration of `%s'
--------	--

[Explanation]

The "long" is coupled with an invalid type specifier.

E4078C	invalid declaration specifier of both fundamental type and elaborate type for declaration of `%s'
--------	---

[Explanation]

Type specifiers of the arithmetic type and of the elaborate type (struct, union or enumeration) are specified simultaneously.

E4079C	invalid declaration specifier of elaborate types for declaration of `%s'
--------	--

[Explanation]

Several type specifiers of the elaborate type (struct, union or enumeration) are specified.

E4080C	invalid struct/union definition: zero size structure
--------	--

[Explanation]

No valid member is declared in a tag of the struct or union. At least one member must be declared.

E4081C	invalid declaration of `%D' as parameter
--------	--

[Explanation]

The declaration of the parameter is invalid.

Remark: This message is not used.

E4082C	redeclaration of `%D' in parameter list
--------	---

[Explanation]

A symbol is redefined in the parameter list.

Remark: This message is not used.

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.

E4089C	invalid storage class specifier `%A' at %S specified for `%D'
--------	---

[Explanation]

The indicated storage class specifier cannot be used in a declaration. Inspect the relation between the specifier and the position it is declared.

E4090C	invalid initializer for declaration of `%D'
--------	---

[Explanation]

An initializer is specified in the declaration where an initialization is invalid.

E4091C	invalid type of array subscript: integral type expected
--------	---

[Explanation]

Only the integral type is allowed for a subscript of the array.

E4092C	both separate parameter declaration and parameter list declaration are used
--------	---

[Explanation]

Though a function is defined with a prototype declaration, non-prototype parameters are also declared. The separate parameter declaration is not needed.

E4093C	invalid constant expression in array subscript
--------	--

[Explanation]

The expression specified in a subscript of an array is invalid. Specify a valid integral constant expression.

E4094C	invalid array subscript: integral constant expression is expected
--------	---

[Explanation]

Only an integral constant expression can be specified as a subscript of the array.
Specify a valid constant expression.

E4095C	subscript of array is negative
--------	--------------------------------

[Explanation]

A negative value is specified as a subscript of the array. Cannot determine the size of the array.

E4096C	array type of functions
--------	-------------------------

[Explanation]

The element of an array is the function type, not a proper type for an element.

E4097C	array type of voids
--------	---------------------

[Explanation]

The element of an array is the void type, not a proper type for an element.

E4098C	function cannot return array
--------	------------------------------

[Explanation]

The return value of the function is the array type. A function cannot return an array.

E4099C	function cannot return function
--------	---------------------------------

[Explanation]

The return value of the function is the function type. A function cannot return a function.

E4100C	parameter name is not specified
--------	---------------------------------

[Explanation]

A parameter name is needed in the function definition of the prototyped function.

E4101C	invalid storage class specifier `%A' specified for function definition of `%D'
--------	--

[Explanation]

The storage class specifier indicated cannot be used in function definition.

E4102C	function type is required for `%D'
--------	------------------------------------

[Explanation]

In the form of the function declaration where a function body follows a declarator, the symbol must be the function type.

E4103C	different data type declaration of `%D'
--------	---

[Explanation]

The symbol defined as a function is already declared not as a function.

E4104C	redeclaration of parameter `%D'
--------	---------------------------------

[Explanation]

The symbol indicated is already declared as a parameter.

E4105C	parameter `%D' is missing in parameter list
--------	---

[Explanation]

The parameter declared in the function definition without prototype declaration is not found in the list of identifiers in the function declarator.

E4106C	invalid void type parameter declaration in parameter list
--------	---

[Explanation]

The void type is specified at the second or latter parameter. The type "void" means that the function takes no parameter, so it must be specified solely with no parameter name.

E4107C	invalid parameter declaration after void type in parameter list
--------	---

[Explanation]

Though the first parameter in the parameter list is declared as the void type, another parameter follows. The "void" parameter means that the function takes no parameter.

E4108C	duplicate member `%I' in `%T'
--------	-------------------------------

[Explanation]

The symbol indicated is already declared as a member of the struct or union.

E4109C	integral constant expression required for enumerator initialization of `%I'
--------	---

[Explanation]

Specify an integral constant as an initializer of the enumerator.

E4110C	undeclared identifier `%I'
--------	----------------------------

[Explanation]

The identifier used in the expression is not declared. An identifier must be declared before using it in an expression except for function call.

E4111C	bit-field size of `%D' is not integral constant expression
--------	--

[Explanation]

The width of a bit-field must be specified by an integral constant expression.

E4112C	type of bit-field member `%D' is not integral type
--------	--

[Explanation]

A bit-field member must be the integral type.

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).

E4119C	invalid operands: %s: %O
--------	--------------------------

[Explanation]

An invalid operand is specified. This message is built in E4001C, E4002C, E4005C, E4006C, E4007C, and E4008C.

Remark: This message is not used.

E4120C	%Z: invalid lvalue
--------	--------------------

[Explanation]

The specified expression cannot be an lvalue (an expression as a destination of the assignment operation).

E4121C	%Z: invalid lvalue: void type specified
--------	---

[Explanation]

An expression of the void type cannot be an lvalue (an expression as a destination of the assignment operation).

E4122C	%Z: invalid lvalue: function type specified
--------	---

[Explanation]

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).

E4123C	%Z: invalid lvalue: array type specified
--------	--

[Explanation]

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).

E4124C	%Z: pointer type or array type required: %O
--------	---

[Explanation]

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.

E4125C	%Z: different struct/union types
--------	----------------------------------

[Explanation]

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.

E4126C	void type function `%D' cannot return void
--------	--

[Explanation]

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.

E4127C	%Z: void type specified: `%D'
--------	-------------------------------

[Explanation]

The void type is used in the context where the type is not allowed. For example, in an initialization.

E4128C	%Z: void type specified: argument %d
--------	--------------------------------------

[Explanation]

A void type expression is specified as an argument of a function call. The void type cannot be used in an expression.

E4129C	%Z: void type specified: %O
--------	-----------------------------

[Explanation]

The void type is used in the context where the type is not allowed. For example, in an assignment.

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.

Remark: This message is not used.

E4136C	%Z: %s: `%T'
--------	--------------

[Explanation]

Cannot determine the size. This message is built in E4009C, E4010C, and E4014C.

Remark: 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 <code>`%T'</code> is undefined in <code>`%T': %O</code>
--------	---

[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.

E4142C	invalid type conversion to array type
--------	---------------------------------------

[Explanation]

It is invalid to convert to the array type.

E4143C	invalid type conversion to different non scalar type
--------	--

[Explanation]

The conversion to a different type of a struct or union is invalid.

E4144C	invalid type conversion to <code>`%T'</code>
--------	--

[Explanation]

Types cannot be converted to each other.

E4145C	invalid type conversion to array type
--------	---------------------------------------

[Explanation]

The conversion from the array type to the struct or union type is invalid.

E4146C	invalid type conversion to function type
--------	--

[Explanation]

The conversion from the function type to the struct or union type is invalid.

E4147C	invalid type for conditional expression: '%T' specified: scalar type expected: %O
--------	---

[Explanation]

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.

E4148C	duplicate case value %X in switch clause
--------	--

[Explanation]

A value specified in a case label must be unique in a switch statement.

E4149C	duplicate 'default' in switch clause
--------	--------------------------------------

[Explanation]

The default label can be used only once in a switch statement.

E4150C	integral type required for switch conditional expression: '%T' specified
--------	--

[Explanation]

Only the integral type can be specified in a condition expression of a switch statement.

E4151C	void type expression is used as argument: argument %d
--------	---

[Explanation]

The expression of a void type argument is specified corresponding to the ellipsis in the parameter list in the function declaration. The information of the function is issued in E4214C or E4215C.

E4152C	too many arguments to function: argument %d expect %d
--------	---

[Explanation]

More arguments are specified than parameters declared for the function. The information of the function is issued in E4214C or E4215C.

E4153C	too few arguments to function: argument %d expect %d
--------	--

[Explanation]

Less arguments are specified than parameters declared for the function. The information of the function is issued in E4214C or E4215C.

Remark: This message is not used.

E4154C	cannot take address of bit-field '%D'
--------	---------------------------------------

[Explanation]

The expression to show a bit-field is specified as an operand of & operator. The address of a bit-field is not available.

E4155C	argument passing to function '%N': argument %d
--------	--

[Explanation]

The argument is passed to the function.

Remark: This message is not used.

E4156C	integral type required for case expression: '%T' specified
--------	--

[Explanation]

Only an integral type can be specified in the case label as in the condition expression of the switch statement.

E4157C	integral constant expression required for case expression
--------	---

[Explanation]

Only the integral constant can be specified in the case label expression.

E4158C	incompatible types between `%T' and `%T'
--------	--

[Explanation]

Types are incompatible in the redeclaration of the symbol by the external reference or in the assignment operation of a struct.

E4159C	redefinition of %s `%D'
--------	-------------------------

[Explanation]

The symbol is doubly defined.

E4160C	%Z: void type specified: argument %d of `%D'
--------	--

[Explanation]

The expression for a void type argument is specified in the function call. The type is not allowed in an expression.

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.

Remark: This message is not used.

E4177C	too many same data types specified for declaration of '%s'
--------	--

[Explanation]

The same type specifiers appear in a declaration.

E4178C	invalid void type parameter before ellipsis in parameter list
--------	---

[Explanation]

The ellipsis cannot be specified with void in the parameter declaration.

E4179C	different kind tag declaration of '%D'
--------	--

[Explanation]

The tag name of the struct, union or enumeration is already declared as a different kind tag.

E4180C	width of type is too wide: cannot use '%T' for bit-field: declaration of '%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.

Remark: This message is not used.

E4181C	invalid null subscript of array: '%D'
--------	---------------------------------------

[Explanation]

The subscript of the array is not specified. Cannot determine the size of the symbol.

E4182C	%Z: `%T' is incomplete type
--------	-----------------------------

[Explanation]

The type is incomplete (the struct, union or enumeration whose tag is 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.

Remark: This message is not used.

E4185C	%Z: missing argument
--------	----------------------

[Explanation]

This is an invalid argument specification for the built-in function.

Remark: This message is not used.

E4186C	%Z: too many arguments
--------	------------------------

[Explanation]

This is an invalid argument specification for the built-in function.

Remark: This message is not used.

E4187C	%Z: invalid argument: pointer required
--------	--

[Explanation]

This is an invalid argument specification for the built-in function.

Remark: This message is not used.

E4188C	invalid type conversion to function
--------	-------------------------------------

[Explanation]

Cannot convert to the function type.

E4189C	automatic symbol has zero or negative size
--------	--

[Explanation]

An automatic variable has zero or negative size because of an invalid declaration of a tag or a subscript of an array.

E4190C	invalid typedefed function type for function definition
--------	---

[Explanation]

A typedef name of the function type cannot be used in the function definition.

E4191C	cannot quote EOF
--------	------------------

[Explanation]

The EOF is found just after the backslash in a character string or character constant.

E4192C	invalid '\W' in input
--------	-----------------------

[Explanation]

A backslash is found outside a character string or character constant. It is an invalid character in a source file.

E4193C	asm: invalid type '%T' of symbol '%D'
--------	---------------------------------------

[Explanation]

A symbol not of the array, arithmetic nor pointer type is specified in the LOAD/STORE notation in the asm statement.

E4194C	asm: array '%D' cannot be used for store operand
--------	--

[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.

E4195C	asm: too many operands
--------	------------------------

[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.

E4196C	asm: invalid expression: '%I'
--------	-------------------------------

[Explanation]

An invalid symbol is specified in the LOAD/STORE notation in the asm statement.

E4197C	asm: regular string constant expected for asm statement
--------	---

[Explanation]

The operand of the asm statement must be a simple string literal.

E4198C	asm: identifier is expected
--------	-----------------------------

[Explanation]

No identifier is specified in the LOAD/STORE notation in the asm statement.

E4199C	asm: '%c' is expected
--------	-----------------------

[Explanation]

The identifier is not enclosed with '}' nor ']' in the LOAD/STORE notation in the asm statement.

E4200C	asm: typedefed identifier '%I' cannot be used in asm statement
--------	--

[Explanation]

The identifier specified in the LOAD/STORE notation in the asm statement is a typedef name.

E4201C	asm: invalid assignment of enumerator `%I'
--------	--

[Explanation]

The identifier specified in the STORE notation in the asm statement is an enumeration constant. Cannot "STORE" to an enumeration constant.

E4202C	asm: invalid expression
--------	-------------------------

[Explanation]

An invalid string literal is found in the asm statement.

E4203C	asm: invalid use of enumerator `%I'
--------	-------------------------------------

[Explanation]

The identifier specified in the LOAD notation in the asm statement is an enumeration constant. Cannot "LOAD" an enumeration constant.

E4204C	`default' outside switch
--------	--------------------------

[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.

E4205C	sorry, internal limitation: number of array dimensions exceeds %d
--------	---

[Explanation]

The dimension of an array exceeds the internal maximum value of the compiler.

E4206C	%Z: cannot get alignment of `%T'
--------	----------------------------------

[Explanation]

Cannot get boundary alignment of an incomplete type.

Remark: This message is not used.

E4207C	invalid type combination of ``%T' and ``%T': %O
--------	---

[Explanation]

The combination of types for operands is invalid.

E4208C	invalid type combination of ``%T' and ``%T': integral type required: %O
--------	---

[Explanation]

The combination of types for operands is invalid. An operand of the integral type is needed.

E4209C	invalid type combination of ``%T' and ``%T': arithmetic type required: %O
--------	---

[Explanation]

The combination of types for operands is invalid. An operand of the arithmetic type is needed.

E4210C	invalid type combination of ``%T' and ``%T': void type required: %O
--------	---

[Explanation]

The combination of types for operands is invalid. An operand of the void type is needed.

E4211C	invalid type combination of ``%T' and ``%T': compatible types required: %O
--------	--

[Explanation]

The combination of types for operands is invalid. An operand of a compatible type is needed.

E4212C	only one argument is specified
--------	--------------------------------

[Explanation]

The "_regload" and "_regstore" require two arguments.

E4213C	too many argument specified
--------	-----------------------------

[Explanation]

The "_regload" and "_regstore" require two arguments.

E4214C	argument passing to builtin function `%N'
--------	---

[Explanation]

This is an invalid usage of the built-in function. An information of the function related to E4151C, E4152C, E4153C, or E4175C.

E4215C	argument passing to function `%N': \"%s\", line %
--------	---

[Explanation]

This is an invalid usage of the function. An information of the function related to E4151C, E4152C, E4153C, or E4175C.

E4216C	invalid declaration specifier: `%s'
--------	-------------------------------------

[Explanation]

The declaration specifier is invalid.

Remark: This message is not used.

E4217C	member `%D' has invalid storage class specifier
--------	---

[Explanation]

A storage class specifier is invalid in the declaration of the member for the struct or union.

Remark: This message is not used.

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.

E4224C	#pragma intvect: the same vector number is expected
--------	---

[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.

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".

E4236C	#pragma register: invalid constant for register bank number
--------	---

[Explanation]

An invalid integral constant is specified as a register number in the "#pragma register".

E4237C	#pragma ilm: invalid constant for interrupt level value
--------	---

[Explanation]

An invalid integral constant is specified as an interrupt level in the "#pragma ilm".

E4238C	pointer to struct/union type required for left operand: access to member `%I': %O
--------	---

[Explanation]

A type other than the pointer type to a struct or union is specified on the left-hand side of the "-%" operator.

E4239C	struct/union type required for left operand: access to member `%I': %O
--------	--

[Explanation]

A type other than struct or union is specified on the left-hand side of the operator.

E4240C	#pragma ilm: interrupt level number should be an integer value without the sign
--------	---

[Explanation]

An interrupt level number is specified badly for the "#pragma ilm". It must be an integer value with no sign specified.

E4241C	invalid restrict qualifier
--------	----------------------------

[Explanation]

The type qualifier "restrict" is effective only on an array or a pointer type.

E4242C	invalid type qualifier in array
--------	---------------------------------

[Explanation]

The type qualifier which is applicable to a subscript of an array is the "restrict" only.

E4243C	sorry, internal limitation: stack size exceeds 32768 bytes
--------	--

[Explanation]

The stack size cannot exceed 32768 bytes in a function.

E4244C	array is too large
--------	--------------------

[Explanation]

The array size cannot be expressed by the size_t type.

E4245C	#pragma section: align cannot be specified for INTVECT section.
--------	---

[Explanation]

#pragma section: align cannot be specified for INTVECT section.

E4246C	#pragma %s: align and locate cannot be specified at the same time.
--------	--

[Explanation]

#pragma %s: align and locate cannot be specified at the same time.

E4247C	enum type cannot be specified for type of bit-field member '%D'
--------	---

[Explanation]

enum type cannot be specified for type of bit-field member '%D'

E4248C	cannot define enumeration tag: `%E %I'
--------	--

[Explanation]

cannot define enumeration tag: `%E %I'

F9001C	too many errors
--------	-----------------

[Explanation]

Too many errors are detected to continue.

F9002C	detected too many error to terminate compilation
--------	--

[Explanation]

Too many syntax errors are detected to continue.

F9003C	too large message file
--------	------------------------

[Explanation]

This is invalid message file.

F9004C	too many messages
--------	-------------------

[Explanation]

This is invalid message file.

F9005C	illegal message file
--------	----------------------

[Explanation]

This is invalid message file.

F9006C	virtual memory exhausted
--------	--------------------------

[Explanation]

More working memory area is needed.

F9007C	data file : cannot open %s
--------	----------------------------

[Explanation]

data file : cannot open %s

F9008C	data file : file read error
--------	-----------------------------

[Explanation]

data file : file read error

F9009C	data file : illegal format %s
--------	-------------------------------

[Explanation]

data file : illegal format %s

F9010C	data file : too many characters of line
--------	---

[Explanation]

data file : too many characters of line

F9011C	data file : mismatch <F>-</F>
--------	-------------------------------

[Explanation]

data file : mismatch <F>-</F>

F9012C	data file : common information cannot describe the multiple.
--------	--

[Explanation]

data file : common information cannot describe the multiple.

APPENDIX F Major Changes

Page	Section	Change Results
Revision 4.1		
-	-	Company name and layout design change
Revision 5.0		
33	● -V, -XV	Deleted Output Example.
43	● -K EOPT, -K NOEOPT	Described effective optimization level is grater than 1 or equal.
43	● -K LIB, -K NOLIB	Described effective optimization level is grater than 1 or equal.
44	● -K NOALIAS, -K ALIAS	Described effective optimization level is grater than 1 or equal.
45	● -x function name 1 [, function name 2, ...], -Xx	Described effective optimization level is grater than 1 or equal.
45	● -xauto [size], -Xxauto	Described effective optimization level is grater than 1 or equal.
45	● -K ARRAY, -K NOARRAY	Described effective optimization level is grater than 1 or equal.



INDEX

Symbols

__wait_nop	
__wait_nop Built-in Function	91
_abort	
_abort Function	110
_exit	
_exit Function	109

A

abort	
_abort Function	110
Acceptable Comment Entry	
Acceptable Comment Entry in Option File	52
Access	
direct Area Access Function	83
I/O Area Access Function	82
Addition	
Arithmetic Operation	
(Addition, Subtraction, Multiplication, and Division)	115
ANSI Standard	
Macros Stipulated by ANSI Standard	92
Area Management	
Area Management	120
Argument	
fcc896s Command Argument	67
fcc896s Command Argument Extension Format ...	68
Arithmetic Operation	
Arithmetic Operation	
(Addition, Subtraction, Multiplication, and Division)	115
asm Statement	
Description by asm Statement	76

B

Basic Process	
fcc896s Command Basic Process	3
Bit Field	
fcc896s Command Bit Field	62
Boundary Alignment	
fcc896s Command Boundary Alignment	61

C

-c	
-c	27
c	
-c	27
C Compiler	
C Compiler Functions	2
C Library	
About Reentrancy of the C Library Functions	134
Calling Procedure	
fcc896s Command Calling Procedure	69
Cancel	
List of Command Cancel Options	23

Char	
The Type of the Truth-value of the Operation Result has Type Char	133
Command	
About the Low-level Library for the Simulator	
Debugger in the fcc896s Command	121
Command Line	16
Command Operands	17
Command Process	16
Command Related Options	25, 48
fcc896s Command Argument	67
fcc896s Command Argument Extension	
Format	68
fcc896s Command Basic Process	3
fcc896s Command Bit Field	62
fcc896s Command Boundary Alignment	61
fcc896s Command Calling Procedure	69
fcc896s Command Dynamic Allocation Area	
Change	123
fcc896s Command Function Call Interface	65
fcc896s Command Interrupt Function Call Interface	
.....	72
fcc896s Command Register Guarantee	70
fcc896s Command Register Setup	70
fcc896s Command Return Value	71
fcc896s Command sbrk.c Source Program	
List	123
fcc896s Command Section Structure	58
fcc896s Command Stack Frame	66
fcc896s Command Structure/Union	64
List of Command Cancel Options	23
List of Command Options	20
Macros Predefined by fcc896s Command	92
Position within Command Line	19
Comment Entry	
Acceptable Comment Entry in Option File	52
Comparison	
Comparison	116
Compiler	
C Compiler Functions	2
Compiler-dependent Language Specification	
Differentials	112
Generation Rules for Names Used by Compiler	60
Compiler Translation	
Limitations on Compiler Translation	93
Compiler-dependent Language	
Compiler-dependent Language Specification	
Differentials	112
Compiling	
Options for Compiling Process Control	3
Conversion	
Type Conversion (Floating-point Number ->Floating- point Number)	117
Type Conversion	
(Floating-point Number ->Integer)	116
Type Conversion	
(Integer ->Floating-point Number)	116
Coordination	
Coordination with Symbolic Debugger	5
ctype	
ctype.h	126
D	
Data Output	
Data Output Related Options	25, 31
Debug Information	
Debug Information Related Options	25, 47
Debugger	
About the Low-level Library for the Simulator	
Debugger in the fcc896s Command	121
Coordination with Symbolic Debugger	5
Default Argument Promotions	
The Default Argument Promotions are not Applied	133
Default Option File	
Default Option File	53
Dependency	
Exclusiveness and Dependency	19
direct Area Access	
direct Area Access Function	83
Directory Names	
File Names and Directory Names	18
Division	
Arithmetic Operation	
(Addition, Subtraction, Multiplication, and Division)	115
Dynamic Allocation	
fcc896s Command Dynamic Allocation Area	
Change	123
E	
-E	
-E	26
E	
-E	26
Enumeration Type	
The Type of the Enumeration Type has a Minimum Type	132
errno	
errno.h	126
Error	
Error Level	55
Error Message	
Format of Error Message	136
Example	
Example	121

Exclusiveness	
Exclusiveness and Dependency	19
Execution Process	
Execution Process Overview	96
exit	
_exit Function	109
F	
fcc896s	
About the Low-level Library for the Simulator	
Debugger in the fcc896s Command	121
fcc896s Command Argument	67
fcc896s Command Argument Extension	
Format	68
fcc896s Command Basic Process	3
fcc896s Command Bit Field	62
fcc896s Command Boundary Alignment	61
fcc896s Command Calling Procedure	69
fcc896s Command Dynamic Allocation Area	
Change	123
fcc896s Command Function Call Interface	65
fcc896s Command Interrupt Function Call	
Interface	72
fcc896s Command Register Guarantee	70
fcc896s Command Register Setup	70
fcc896s Command Return Value	71
fcc896s Command sbrk.c Source Program	
List	123
fcc896s Command Section Structure	58
fcc896s Command Stack Frame	66
fcc896s Command Structure/Union	64
Macros Predefined by fcc896s Command	92
FELANG	
FELANG	13
FETOOL	
FETOOL	8
File Names	
File Names and Directory Names	18
File Organization	
File Organization	100
float	
float.h	126
Floating-point Data	
Floating-point Data Format and Expressible Value	
Range	114
Floating-point Number	
Type Conversion (Floating-point Number -> Floating-point Number)	117
Type Conversion	
(Floating-point Number -> Integer)	116
Type Conversion	
(Integer -> Floating-point Number)	116
Function Call Interface	
fcc896s Command Function Call Interface	65
G	
Generation Rules	
Generation Rules for Names Used by Compiler	60
H	
Header File	
Header File Search	5
I	
I/O Area Access	
I/O Area Access Function	82
Identifier	
Tool Identifier	54
INC896	
INC896	11
In-line	
In-line Expansion Specifying Function	84
Instruction	
Description by Pragma Instruction	77
Integer	
Type Conversion	
(Floating-point Number -> Integer)	116
Type Conversion	
(Integer -> Floating-point Number)	116
Integer Constant	
The Type of the Integer Constant has a Minimum	
Type	132
Integral Promotions	
The Integral Promotions are not Applied	131
Interface	
fcc896s Command Function Call Interface	65
fcc896s Command Interrupt Function Call	
Interface	72
Interrupt	
fcc896s Command Interrupt Function Call	
Interface	72
Interrupt Function Calling Procedure	74
Interrupt Function Description Function	80
Interrupt Level Setup Function	79, 89
Interrupt Mask Disable Function	79
Interrupt Mask Setup Function	79
Interrupt Stack Frame	73
Interrupt Vector Table Generation Function	81
No-register-save Interrupt Func. Function	90
Interrupt Level	
Interrupt Level Setup Function	79, 89
Interrupt Vector Table	
Interrupt Vector Table Generation Function	81

L

Language

- Compiler-dependent Language Specification
 - Differentials 112

Language Specification

- Language Specification Related Options 25, 36

LIB896

- LIB896 9

Libraries

- Operations Specific to Libraries 129

Library

- About the Low-level Library for the Simulator
 - Debugger in the fcc896s Command 121
- Library Section Names 101
- Processes and Functions Required for Library
 - Use 104
- Standard Library Functions and Required Process/
 - Low-level Functions 106

Limitations

- Limitations on Compiler Translation 93

limits

- limits.h 126

Linkage

- Linkage Related Options 25, 49

Load Module

- Load Module Creation 121

Low-level Function

- Low-level Function (System-dependent Process)
 - Types 102
- Low-level Function Library Overview 120
- Low-level Function Specifications 107
- Low-level Function Types 105
- Standard Library Functions and Required Process/
 - Low-level Functions 106

Low-level Library

- About the Low-level Library for the Simulator
 - Debugger in the fcc896s Command 121

M

Macros

- Macros Predefined by fcc896s Command 92
- Macros Stipulated by ANSI Standard 92

Mask

- Interrupt Mask Disable Function 79
- Interrupt Mask Setup Function 79

math

- math.h 127

Messages

- Messages Generated in Translation Process 54

Minimum Type

- The Type of the Enumeration Type has a Minimum
 - Type 132

- The Type of the Integer Constant has a Minimum

- Type 132

Multiple Specifying

- Multiple Specifying of Same Option 19

Multiplication

- Arithmetic Operation
 - (Addition, Subtraction, Multiplication, and Division) 115

N

No-register-save

- No-register-save Interrupt Func. Function 90

O

Operands

- Command Operands 17

OPT896

- OPT896 10

Optimization

- Optimization 6
- Optimization Related Options 25, 41

Option

- Command Related Options 25, 48
- Data Output Related Options 25, 31
- Debug Information Related Options 25, 47
- Language Specification Related Options 25, 36
- Linkage Related Options 25, 49
- List of Command Cancel Options 23
- List of Command Options 20
- Multiple Specifying of Same Option 19
- Optimization Related Options 25, 41
- Option File Related Options 25, 51
- Option Syntax 19
- Options for Compiling Process Control 3
- Output Object Related Options 25, 46
- Preprocessor Related Options 25, 28
- Translation Control Related Options 25, 26

Option File

- Acceptable Comment Entry in Option File 52
- Default Option File 53
- Option File 52
- Option File Related Options 25, 51

Output Object

- Output Object Related Options 25, 46

P

-P

- P 26

P

- P 26

Pragma Instruction

- Description by Pragma Instruction 77

Preprocessor	
Preprocessor Related Options	25, 28
R	
Reentrancy	
About Reentrancy of the C Library	
Functions	134
Register Bank	
Register Bank Number Setup Function	88
Register Guarantee	
fcc896s Command Register Guarantee	70
Register Setup	
fcc896s Command Register Setup	70
Return Value	
fcc896s Command Return Value	71
S	
-S	
-S	27
S	
-S	27
sbrk	
fcc896s Command sbrk.c Source Program	
List	123
sbrk Function	108, 122
Search	
Header File Search	5
Section Name	
Library Section Names	101
Section Name Change Function	85
Section Structure	
fcc896s Command Section Structure	58
Sensitiveness	
Case Sensitiveness	19
setjmp	
setjmp.h	128
Simulator Debugger	
About the Low-level Library for the Simulator	
Debugger in the fcc896s Command	121
Source Program	
fcc896s Command sbrk.c Source Program	
List	123
Stack Frame	
fcc896s Command Stack Frame	66
Interrupt Stack Frame	73
Standard Library Function	
Standard Library Functions and Required Process/	
Low-level Functions	106
Startup Routine	
Startup Routine Creation	98
Statement	
Description by asm Statement	76
stdarg	
stdarg.h	127
stddef	
stddef.h	127
stdio	
stdio.h	127
stdlib	
stdlib.h	127
string	
string.h	128
Structure	
fcc896s Command Structure/Union	64
Subtraction	
Arithmetic Operation	
(Addition, Subtraction, Multiplication, and	
Division)	115
Symbolic Debugger	
Coordination with Symbolic Debugger	5
Syntax	
Option Syntax	19
System-dependent Process	
Low-level Function (System-dependent Process)	
Types	102
System-dependent Processes	102
T	
TMP	
TMP	12
Tool	
Tool Identifier	54
Translation Control	
Translation Control Related Options	25, 26
Translation Process	
Messages Generated in Translation Process	54
Truth-value	
The Type of the Truth-value of the Operation Result	
has Type Char	133
Type Conversion	
Type Conversion (Floating-point Number -> Floating-	
point Number)	117
Type Conversion	
(Floating-point Number -> Integer)	116
Type Conversion	
(Integer -> Floating-point Number)	116
U	
Union	
fcc896s Command Structure/Union	64
W	
wait_nop	
__wait_nop Built-in Function	91

CM25-00202-5E

SpanSion • SOFTWARE SUPPORT MANUAL

F²MC-8L/8FX FAMILY
8-BIT MICROCONTROLLER
SOFTUNE™ C COMPILER MANUAL

September 2014 Rev. 5.0

Published **SpanSion Inc.**
Edited **Communications**

Colophon

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 any use that includes 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 any use where chance of failure is intolerable (i.e., submersible repeater and artificial satellite). Please note that Spansion will not be liable to you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products. 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 US Export Administration Regulations or the applicable laws of any other country, the prior authorization by the respective government entity will be required for export of those products.

Trademarks and Notice

The contents of this document are subject to change without notice. This document may contain information on a Spansion product under development by Spansion. Spansion reserves the right to change or discontinue work on any product without notice. The information in this document is provided as is without warranty or guarantee of any kind as to its accuracy, completeness, operability, fitness for particular purpose, merchantability, non-infringement of third-party rights, or any other warranty, express, implied, or statutory. Spansion assumes no liability for any damages of any kind arising out of the use of the information in this document.

Copyright © 2004-2014 Spansion All rights reserved. Spansion®, the Spansion logo, MirrorBit®, MirrorBit® Eclipse™, ORNAND™ and combinations thereof, are trademarks and registered trademarks of Spansion LLC in the United States and other countries. Other names used are for informational purposes only and may be trademarks of their respective owners.