

TOSHIBA

TLCS-900 Assembler Reference

1st Edition

TOSHIBA Corporation Semiconductor Company

RESTRICTIONS ON PRODUCT USE

- The information contained herein is subject to change without notice. (W11AE-01)
- TOSHIBA is continually working to improve the quality and reliability of its products. Nevertheless, the hardware and/or software incorporated in the TOSHIBA products listed in this document ("TOSHIBA Products") in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress. It is the responsibility of the customer, when utilizing TOSHIBA Products, to fully comply with the standards of safety in making safety design for the entire system, and to avoid the situations in which a malfunction or failure of such TOSHIBA Products could cause loss of human life, bodily injury or damage to property.
In developing your designs, please ensure that TOSHIBA Products are used within specified operating ranges as set forth in the specifications for this product, the specifications for the semiconductor devices under evaluation, and any other related information. Also, please keep in mind the precautions and conditions set forth in the "TOSHIBA Semiconductor Reliability Handbook" and "Instruction Manual" or "Operation Manual" that accompany this product and any devices connected to this product.
Please always confirm the latest information of the TOSHIBA Products released on the web page of microcomputer in the web site of TOSHIBA Semiconductor Company.
(<http://www.semicon.toshiba.co.jp/eng/>) (W01AE-01)
- The TOSHIBA Products are intended for usage in the functional evaluation of semiconductor devices. TOSHIBA Products shall not be used for purposes other than functional evaluation, such as for verification of device reliability. The TOSHIBA Products shall not be incorporated this product into customer products. The TOSHIBA Products shall not be converted, disassembled, modified, or used outside its specified operating range of the TOSHIBA Products listed in this document.
- The TOSHIBA Products are intended for the functional evaluation of semiconductor devices that are designed for use in general electronics applications (e.g., computer, personal equipment, office equipment, measuring equipment, industrial robotics, and domestic appliances). These TOSHIBA Products are neither intended nor warranted for usage in equipment that requires extraordinarily high quality and/or reliability or a malfunction or failure of which may cause loss of human life or bodily injury ("Unintended Usage").
Without limiting the generality of the foregoing, unintended Usage include atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, medical instruments, and all types of safety devices. The TOSHIBA Products shall not be used for Unintended Usage. (W02BE-01)
- The products described in this document shall not be used or embedded to any downstream products of which manufacture, use and/or sale are prohibited under any applicable laws and regulations. (W03AE-01)
- TOSHIBA does not take any responsibility for incidental damage (including loss of business profit, business interruption, loss of business information, and other pecuniary damage) arising out of the use or disability to use the product. (W04AE-01)
- The information contained herein is presented only as a guide for the applications of our products. No responsibility is assumed by TOSHIBA for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patents or other rights of TOSHIBA or the third parties. (W06AE-02)
- Product names mentioned herein may be trademarks of their respective companies. (W07AE-02)

Preface

Thank you for using Toshiba microcomputer products.

This manual describes how to use the microcomputer development system product you have purchased. Please keep this manual to hand when you use the product.

Toshiba will continue to make every effort to improve our products to better meet the needs of our customers. We will highly appreciate your continued patronage of Toshiba microcomputer products also in future.

- Microsoft®, Windows®, Windows® 2000, Windows® XP, and Windows Vista® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Technical support

The "readme.txt" file is included with the product package to help you use this product. If you have any further questions regarding the content of this manual, please do not hesitate to contact your local Toshiba sales representative.

Our technical support service is available if you encounter any phenomenon that seems to be faulty while using this product. At your request we will investigate the cause of the phenomenon and report back to you. To use this service, you need to provide us with the data that enables us to reproduce the phenomenon, such as the operation procedure, etc. Please note that we may not be able to deal with a phenomenon that cannot be reproduced.

INDEX

Part 1	About this book	1
Chapter 1	Explanation of this manual.....	3
Part 2	The Assembler.....	5
Chapter 2	The Assembler	7
2.1	Input and Output Files	7
Chapter 3	Assembly Language	8
3.1	Basic Assembler Syntax.....	8
3.1.1	Character Set	8
3.1.2	Reserved Words.....	8
3.1.3	Numerical Value	8
3.1.4	Source Statements	10
3.1.5	Comments	11
3.1.6	Location Counter	11
3.2	Identifier of Assembly Language.....	11
3.2.1	Identifier	11
3.2.2	Types of Identifier	11
3.3	Expression of Assembly Language	12
3.3.1	Expressions.....	12
3.3.2	Operators.....	12
3.4	Machine Instructions in Assembly Language	16
3.4.1	Numeric Range.....	17
3.5	Assembler Directives.....	17
3.5.1	Defining Modules	17
3.5.2	Defining Sections	18
3.5.3	Directives for Inter-modular Identifier Referencing.....	19
3.5.4	Data Area Definition Directives	20
3.5.5	Equ Directives	23
3.5.6	Align Directives	23
3.5.7	Org Directives.....	24
3.6	Control Instructions	24
3.6.1	File Include Function	24
3.6.2	\$maximum Control Instruction	25
Chapter 4	Assembler List File Format	26
Chapter 5	Assembler Options	28
-I	Specify Search Path for Include Files	28
-J	Recognize the Kanji Code (Japanese version only)	28
-Nb	Select CPU Type.....	28
-O	Specify Optimization Level	29
-V	Output Version Number	29
-XE	Ignore Escape Sequence	29
-e	Create Error List File	30
-f	Read Option List File	30
-g	Create Debugger Information at assemble phase.....	30
-l	Create an Assembler List File	30
-o	Specify an Output Filename	31
-w	Specify Warning Level	31
Chapter 6	Assembler Limitation.....	32
Part 3	The Linker	33

Chapter 7	The Linker	35
7.1	Input and Output Files	35
Chapter 8	Link Command File	36
8.1	Basic Link Command File	36
8.1.1	Reserved Words	36
8.1.2	Identifiers	36
8.1.3	Expressions	36
8.1.4	Location Counter	37
8.1.5	Operators and Their Order of Precedence	37
8.1.6	Functional Operators	37
8.1.7	Assign Statements	38
8.1.8	Comments	39
8.2	Memory Definition Part	39
8.2.1	Function of Memory Definition Part	39
8.2.2	Memory Definition Part Format	39
8.2.3	Predefined Memory	40
8.3	Section Definition Part	41
8.3.1	Function of Section Definition Part	41
8.3.2	Section Definition Part Format	41
8.3.3	Output Section Name Field	41
8.3.4	Output Specification Field	42
8.3.5	Attribute Field	43
8.3.6	Input Section Specification Field	45
8.3.7	Padding Field	45
8.3.8	Output Memory Field	45
8.4	Symbol Definition Part	46
8.5	Incremental Linking	46
8.5.1	Processing in Incremental Linking	46
Chapter 9	Linker Map File Format	48
Chapter 10	Linker Options	50
-F	Specify Fill Value for Empty Area in Output Section	50
-L	Specify Search Path for Input Files	50
-V	Output Version Number	50
-e	Create Error List File	51
-g	Create Debugger Information at link phase	51
-l	Create a Link Map File	51
-o	Specify an Output Filename	52
-r	Perform Incremental Linking	52
-u	Link Undefined Symbol	52
-w	Specify Warning Level	53
Chapter 11	Linker Limitation	54

Part 4 The Macro Preprocessor.....55

Chapter 12	The Macro Preprocessor	57
12.1	Macro Preprocessor Overview	57
12.2	Input and Output Files	57
Chapter 13	Macro Preprocessor Grammar	58
13.1	Character Set	58
13.2	Character String	58
13.3	Identifier	58
13.4	Constants	59
13.5	Expressions	60
13.6	Trigger Character	60
13.7	Comments	60

13.8	Line Splice.....	60
Chapter 14	Preprocess Functions	61
Chapter 15	Macroprocess Functions	62
15.1	Operators.....	62
15.2	Conditional Macro.....	63
15.3	Replacement Macro	65
15.4	Numerical Macro.....	66
15.5	String Control Macro	67
15.6	Particular Macro.....	71
Chapter 16	Macro Preprocessor Options	72
-D	Define Macro.....	72
-GN	Specify File Name Used in Error Message.....	72
-I	Specify Search Path for Include Files	72
-J	Recognize the Kanji Code (Japanese version only)	73
-U	Disable the Macro Definition	73
-V	Output Version Number	73
-e	Create Error List File	74
-f	Read Option List File.....	74
-g	Create Debugger Information	74
-l	Create a Macro Preprocessor List File.....	74
-o	Specify an Output Filename	75
-s	Define Identifier of Variable Function.....	75
Chapter 17	Macro Preprocessor Limitation	76
Part 5 The Librarian.....		77
Chapter 18	The Librarian	79
18.1	Librarian Overview	79
18.2	Startup command.....	79
18.3	Input and Output Files	79
Chapter 19	Librarian Options	81
-V	Output Version Number.....	81
-d	Delete a Module	81
-e	Create Error List File	81
-f	Read Option List File.....	82
-l	Output Module Identifier Information	82
-r	Create Library File and Register and Update Modules.....	82
-t	Output Module Information	83
Part 6 The Object Converter		85
Chapter 20	The Object Converter.....	87
20.1	Object Converter Overview	87
20.2	Startup command.....	87
20.3	Input and Output Files	87
Chapter 21	Object Converter Options	88
-F	Select Object Format.....	88
-P	Specify Fill Value for Empty Area.....	88
-V	Output Version Number.....	89
-c	Specify a comment	89
-e	Create Error List File	89
-f	Read Option List File.....	89
-l	Output an Object Converter List.....	90
-o	Specify an Output Filename	90
-ra	Specify Object Output Range (address specification).....	90

-rb	Specify Object Output Range (section specification)	91
-----	---	----

Part 7 Error Message.....93

Chapter 22	Error Message Format.....	95
22.1	Types of Error Message	95
22.2	Error Message Format.....	95
Chapter 23	Assembler Error Messages	96
23.1	Assembler Fatal Errors	96
23.2	Assembler Errors.....	97
23.3	Assembler Warning Errors	98
Chapter 24	Linker Error Messages	100
24.1	Linker Fatal Errors.....	100
24.2	Linker Errors	102
24.3	Linker Warning Errors	103
Chapter 25	Macro Preprocessor Error Messages	105
25.1	Macro Preprocessor Fatal Errors.....	105
25.2	Macro Preprocessor Errors	106
25.3	Macro Preprocessor Warning Errors	109
Chapter 26	Librarian Error Messages.....	110
26.1	Librarian Fatal Errors.....	110
26.2	Librarian Errors	111
26.3	Librarian Warning Errors.....	111
Chapter 27	Object Converter Error Messages.....	112
27.1	Object Converter Fatal Errors.....	112
27.2	Object Converter Errors.....	113
27.3	Object Converter Warning Errors	113

Part 1 About this book

Chapter 1 Explanation of this manual

This manual includes specifications relating to assembly language, the linker and link command files, the Macro Preprocessor, the librarian, and the object converter. They are collectively indicated for every tool.

This manual indicates further details more than "TLCS-900 Compiler System User's Guide" to customize for user needs. Additionally, the contents of explanations relating to option, error message, warning message are included. We hope you will make use of your application development.

Format Description Rules

[Format Description Example]

```
#pragma section <Section Type> [<Section Name>]
                [<Displacement>|<Start Address>]
```

#pragma section For commands and options, etc., parts that do not have the enclosure symbols or delimiting symbols described hereafter are noted as is in the actual program.

<Section Type> Specifiers enclosed in < > describe character strings or numerical values specified within < > in the actual program.

[<Section Name>] Specifiers enclosed in [] can be omitted in the actual program.

[<Displacement> | <Start Address>]

For specifiers delimited by " | ", specify one of those items in the actual program.

Memory Space

The memory space for "TLCS-900 Family" is 16M byte in address range from 0x0 to 0xfffff.

The usable section type, displacement, and address range for section directives which is used in assembler source programs are as follows.

Table 1-1 Displacement and Address range

Section type	Displacement	Address range
data	small	0x0 - 0xff
	medium	0x0 - 0xffff
	large	0x0 - 0xfffff
romdata	medium	0x0 - 0xffff
	large	0x0 - 0xfffff
code	medium	0x0 - 0xffff
	large	0x0 - 0xfffff

Part 2 The Assembler

Chapter 2 The Assembler

2.1 Input and Output Files

Input and Output Files

The suffixes used in the names of input and output files of the Assembler are as follows.

Table 2-1 Input and Output files of the Assembler

Suffix	File type	Classification
.asm	Assembly source file	Input
.rel	Relocatable object file	Output
.lst	Assembly List file	Output

- Assembly source file

These files are the assembly language is described.

- Relocatable object file

These files are the result of assembling. These files are the files that complies with the IEEE695 object module format. Because these files are relocatable, it is not possible to obtain final objects without linking them using the Linker.

- Assemble list file

The list files include the assemble list, symbol list, cross-reference list, etc. The "-l" option must be specified for these files to be output.

Chapter 3 Assembly Language

3.1 Basic Assembler Syntax

The assembly language is described by the source statements according to specific rules of notation. This chapter describes the elements that make up assembly language.

3.1.1 Character Set

The following list shows the characters that can be used when writing an assembly source program.

Character set:

English alphabets | decimal digits | graphic characters | new-line space character

Graphic characters:

! " # \$ % & ' () * + , - . / : ; < = > ? [\] ^ _ { | } ~ @

In addition to the characters listed above, other characters can be used in comments. The question mark (?) and pound sign (#) and (@) are used by Macro Preprocessor. Please handle these three characters, with care.

3.1.2 Reserved Words

Reserved words are predefined character strings used by the Assembler. They include directives, mnemonics, and registers, etc. Reserved words do not differentiate between uppercase and lowercase letters.

See each microcomputer's "Data Sheet" for details about machine instructions and registers.

abs	align	code	data	db	dd	dfb
dfd	dfi	dfp	dfw	dl	dp	dw
dsb	dsd	dsl	dsp	dsw	end	equ
extern	fdatad	fdatas	fdatax	file	fimdh	fimdl
fims	fimxh	fimxl	fimxm	include	large	line
medium	module	org	public	romdata	section	sizeof
small	startof	symbol_a	symbol_b	symbol_n	symbol_t	

3.1.3 Numerical Value

Integer

All integer are handled as 32-bit width numerical values within the assembler. When a numerical value that is not expressed as 32-bit width is specified, only the lower 32-bit width of the specified numerical value is handled. The character using integer do not differentiate between uppercase and lowercase letters.

Binary :	Expressed as strings of 0s or 1s starting with 0b or 0y.
Octal :	Expressed as strings of numbers 0 to 7 starting with 0.
Decimal :	Expressed as strings of numbers 0 to 9 starting with other than 0.
Hexadecimal :	Expressed as alphanumeric strings consisting of numbers 0 to 9 and a to f and starting with 0x.

Floating-point Number

Types of Floating-point number

There are three types of single-precision processing 32-bit data, double-precision processing 64-bit data, and extended double-precision processing 80-bit data in the floating-point number. The following describes each type.

Single-precision floating-point number

The single-precision floating type (float type) is conformed with IEEE754 format. The float type consists of 32-bit data including sign 1-bit, exponent 8-bit, and significand 23-bit.

Double-precision floating-point number

The double-precision floating type (double type) is conformed with IEEE754 format. The double type consists of 64-bit data including sign 1-bit, exponent 11-bit, and significand 52-bit.

Extended double-precision floating-point number

The extended double-precision floating type (long double type) is conformed with IEEE754 format. The long double type consists of 80-bit data including signed 1-bit, exponent 15-bit, and significand 64-bit.

Table 3-1 Type of Floating-point number

Format	Type	Size
Single precision	float	32 bits
Double precision	double	64 bits
Extended double precision	long double	80 bits

Description procedures

The description procedures of the floating-point number is conformed with ANSI or ISO/IEC 9899. When the significant part starts from a period (.12, .3e5F, and etc.), a syntax error occurs. (ANSI stated above and ISO/IEC 9899 allow this format.)

The following are description procedures of the floating-point number.

```

Floating-point number :
    <digit-sequence> . <significand digit part> [<exponent part>] [<floating suffix>]
    | <digit-sequence> . [<exponent part>] [<floating suffix>]
    | <digit-sequence> <exponent part> [<floating suffix>]
Exponent part:
    e [+|-] <digit-sequence>
    | E [+|-] <digit-sequence>
Floating suffix :
    f | l | F | L
  
```

The floating-point number has significand part that is followed by an exponent part and a suffix that specifies its type. The components of the significand part are described a digit-sequence representing the whole-number part, followed by a period (.), followed by a digit-sequence representing the fraction part. The components of the exponent part are described an "e" or "E" followed by an exponent consisting of an optionally signed digit sequence. The significand part needs the whole-number part and either a period or a exponent part.

The suffix represent the floating-point number type. When the suffix is omitted, the floating-point number shall be regarded as the double-precision floating type. The suffix is respectively corresponded to the type as follows.

"f" or "F"	: Single-precision floating type	(float type)
No suffix	: Double-precision floating type	(double type)
"l" or "L"	: Extended double-precision floating type	(long double type)

[Caution]

When a floating-point number is described in a place where a integer constant is required, the decimal point and below of that value is truncated, and the value is converted to a 32-bit width integer constant.

Character Constants

Character constants are expressed with enclosing in single-quotes ('). Character constants have a 32-bit width value and are put in order by the described order, from the start of the string. When there are fewer than 4 bytes, the beginning is padded with 00s. The excess part is omitted when a character exceeds 4 bytes.

Escape Sequence

Two kinds of escape sequence can be used for character constants: one is the escape sequence written directly as a value after the " \ ". This is either an octal number of 3 characters maximum in the range 0 to 377 after the " \ " or a two-digit hexadecimal of 0 to ff after " \x ".

The other is one of the following special escape sequences:

\0	0x00 (NUL)	\a	0x07 (Alert, Bell)
\b	0x08 (Backspace)	\f	0x0c (Form feed)
\n	0x0a (Line feed)	\r	0x0d (Carriage return)
\t	0x09 (Horizontal tab)	\v	0x0b (Vertical tab)
\"	0x22 (")	\'	0x27 (')
\?	0x3f (?)	\\	0x5c (\)

3.1.4 Source Statements**Source Statement Description Format**

With an assembler source program, it is necessary to follow a description format as determined below. Statements written in this description format are called source statements.

[Description Format]

[<Label> :] [<Instruction> [; <Comment>]
--

<Label> The label is identifier that starts at the head of a line, and include a colon (:) at the end of the label. The labels show the location of statement.

<Instruction>

The instruction is machine instructions, directives, or control instructions.

<Comment>

The comment starts with a semicolon (;) and extends to the end of the line. The content of a comment is ignored by the Assembler.

3.1.5 Comments

The comment starts with a semicolon (;) and extends to the end of the line. The content of a comment is ignored by the Assembler.

- A semicolon in the character string enclosed in double quotes (") or single quotes (') is not treated as the start of a comment field but is processed as a normal character.
- When using double comment sign which is described two comment signs continuously, the comment does not appear in an assemble list file.

3.1.6 Location Counter

The location counter show the current address. The location counter value is shown by dollar sign (\$).

The location counter is initialized at the head of the section, moreover, it is updated each time a source statement of one line is converted into the object code.

3.2 Identifier of Assembly Language

3.2.1 Identifier

The character string showing numerical value and address is called identifier. The identifier is character string which start with other than digit.

Usable character set:

English alphabets | decimal digits | period | underscore

The assembler distinguishes between the uppercase and lowercase letters which are used for a identifier.

3.2.2 Types of Identifier

Identifier are classified into the following types.

Label	Labels are identifiers which represent the locations of machine instruction and data areas, and are referenced by jump instructions and branch instructions. Only one label can be described on one line. To define multiple labels in one address, define them on independent lines. When referencing a label, specify it without the colon.
Variable name	Variable names are identifiers defined using the data area definition directives.
Module name	Module names are identifiers defined using the "module" directive.
Section name	Section names are identifiers defined using the "section" directive.
EQU name	EQU names are identifiers defined using the "equ" directive.
External identifier	There are two types of external identifiers: external definition identifiers declared with the "public" directive and external reference identifiers declared with the "extern" directive.

3.3 Expression of Assembly Language

3.3.1 Expressions

Expressions are used to specify the operands of machine instructions and directives. Expressions consist of combinations of one or more terms and operators.

Value of an Expression

The values of expressions are treated as signed 32-bit width integers. If an overflow occurs during valuation of expressions, the part which exceeds 32-bit width is ignored.

Absolute Expression

The expression whose value is fixed at assembling is called absolute expression. For example, integers, character constants, and combination of them and operators. The expression whose value is fixed at linking, for example an external reference symbol and etc. is not an absolute expression.

3.3.2 Operators

Expressions can include arithmetic operators, logical operators, and functional operators. The tables below show the respective operator types. In these operators, four rules of operation of the arithmetic operation and functional operators `fims`, `fimdh`, `fimdl`, `fimxh`, `fimxm`, and `fimxl` only can process a floating-point number.

The operation rules including the floating-point number and the function of each functional operator are described hereinafter.

Arithmetic operators

Table 3-2 Arithmetic operators

Operator	Symbol	Notation
Unary minus	-	-term
Add	+	term + term
Subtract	-	term - term
Multiply	*	term * term
Divide *1	/	term / term
Remainder	%	term % term
Right shift *2	>>	term >> term
Left shift *2	<<	term << term

*1) Division by zero results in an error.

*2) The second term in right and left shifts must be a positive number.

Bitwise operators

Table 3-3 Bitwise operators

Operator	Symbol	Notation
Ones complement	~	~ term
Bitwise AND	&	term & term
Bitwise OR		term term
Bitwise exclusive OR	^	term ^ term

Functional operators

Functional operators are operator which has arguments.

Notation of functional operators

- There must be no space between the functional operator and the left parenthesis " (".
- There is no differentiation between uppercase and lowercase letters in functional operators.

Table 3-4 Functional Operators

Operator	Symbol	Notation
Returns the upper 32-bit pattern of the double-precision floating-point number	fimdh	fimdh(<Expression>)
Returns the lower 32-bit pattern of the double-precision floating-point number	fimdl	fimdl(<Expression>)
Returns the bit pattern of the single-precision floating-point number	fims	fims(<Expression>)
Returns the upper 16-bit pattern of the extended double-precision floating-point number	fimxh	fimxh(<Expression>)
Returns the lower 32-bit pattern of the extended double-precision floating-point number	fimxl	fimxl(<Expression>)
Returns the middle 32-bit pattern of the extended double-precision floating-point number	fimxm	fimxm(<Expression>)
Returns the size of the section	sizeof	sizeof(<Section_name>)
Returns the start address of the section	startof	startof(<Section_name>)

fimdh

[Description Format]

```
fimdh( <Expression> )
```

[Function]

For fimdh, the upper 32 bits of a double-precision floating type (double type) constant specified in the <Expression> is returned as a value converted to the IEEE754 format.

[Explanation]

- The evaluation result of <Expression> must be the double-precision floating constant.
- When <Expression> is except double-precision floating-point number, a warning occurs. Thus, <Expression> is regarded as the double-precision floating type.
- When the evaluation result of <Expression> against the description rules of the floating-point number, an error occurs. Thus the value of <Expression> is processed as 0.

fimdl

[Description Format]

```
fimdl( <Expression> )
```

[Function]

For fimdl, the lower 32 bits of a double-precision floating type (double type) constant specified in the <Expression> is returned as a value converted to the IEEE754 format.

[Explanation]

- The evaluation result of <Expression> must be the double-precision floating constant.
- When <Expression> is except double-precision floating-point number, a warning occurs. Thus, <Expression> is regarded as the double-precision floating type.
- When the evaluation result of <Expression> against the description rules of the floating-point number, an error occurs. Thus the value of <Expression> is processed as 0.

fims**[Description Format]**

fims(<Expression>)

[Function]

For fims, the single-precision floating type (float type) constant specified in the <Expression> is returned as a value converted to the IEEE754 format.

[Explanation]

- The evaluation result of <Expression> must be the single-precision floating constant.
- When <Expression> is except single-precision floating-point number, a warning occurs. Thus, <Expression> is regarded as the single-precision floating type.
- When the evaluation result of <Expression> against the description rules of the floating-point number, an error occurs. Thus the value of <Expression> is processed as 0.

fimxh**[Description Format]**

fimxh(<Expression>)

[Function]

For fimxh, the upper 16 bits of a extended double-precision floating type (long double type) constant specified in the <Expression> is returned as a value converted to the IEEE754 format.

[Explanation]

- The evaluation result of <Expression> must be the extended double-precision floating constant.
- When <Expression> is except extended double-precision floating-point number, a warning occurs. Thus, <Expression> is regarded as the extended double-precision floating type.
- When the evaluation result of <Expression> against the description rules of the floating-point number, an error occurs. Thus the value of <Expression> is processed as 0.

fimxl**[Description Format]**

fimxl(<Expression>)

[Function]

For fimxl, the lower 32 bits of a extended double-precision floating type (long double type) constant specified in the <Expression> is returned as a value converted to the IEEE754 format.

[Explanation]

- The evaluation result of <Expression> must be the extended double-precision floating constant.

- When <Expression> is except extended double-precision floating-point number, a warning occurs. Thus, <Expression> is regarded as the extended double-precision floating type.
- When the evaluation result of <Expression> against the description rules of the floating-point number, an error occurs. Thus the value of <Expression> is processed as 0.

fimxm

[Description Format]

```
fimxm(<Expression>)
```

[Function]

For fimxm, the middle 32 bits (from 17th to 48th) of a extended double-precision floating type (long double type) constant specified in the <Expression> is returned as a value converted to the IEEE754 format.

[Explanation]

- The evaluation result of <Expression> must be the extended double-precision floating constant.
- When <Expression> is except extended double-precision floating-point number, a warning occurs. Thus, <Expression> is regarded as the extended double-precision floating type.
- When the evaluation result of <Expression> against the description rules of the floating-point number, an error occurs. Thus the value of <Expression> is processed as 0.

sizeof

[Description Format]

```
sizeof(<Section_name>)
```

[Function]

Returns the size of the section specified in <Section_name>.

[Explanation]

- The section size returned by sizeof does not fix until all files have been linked.
- Specify a section name defined by a section directive in that module in <Section_name>. An error occurs if the specified section name does not exist in that module.

startof

[Description Format]

```
startof(<Section_name>)
```

[Function]

Returns the start address of the section specified in <Section_name>.

[Explanation]

- The start address returned by startof does not fix until all files have been linked.
- Specify a section name defined by a section directive in that module in <Section_name>. An error occurs if the specified section name does not exist in that module.

Operator Precedence

The operators have the order of precedence in using. The tables below shows the operator precedence.

Table 3-5 Operator Precedence

Precedence	Operator
1	Unary operators (- , ~) and functional operators (sizeof, sizeof, etc.)
2	Multiply, divide, and remainder (* , / , %)
3	Add and subtract (+ , -)
4	Shift operations (>> , <<)
5	Bitwise AND (&)
6	Bitwise Exclusive OR (^)
7	Bitwise OR ()

3.4 Machine Instructions in Assembly Language

Machine instruction are instructions for the microcomputer to actually operate. The description format follow the source statement description format, <Instruction> field is described as follows.

[Description Format]

```
[ <Label>: ]
    <Mnemonic> [ <Operand> ] [ ; <Comment> ]
```

<Mnemonic> It is a name of a machine instruction.

<Operand> <Operand> is specified pre-established addressing mode by each <Mnemonic>. The addressing mode vary each microcomputer.

See each microcomputer's "Data Sheet" for details about machine instructions. In addition, some instructions are converted to situational machine instruction in assembler language, and they are undocumented in "Data Sheet".

- Jump instruction (j)

A jump instruction is converted to the smallest machine instruction, considering the range to branch destination, displacement, absolute branch or relative branch, when the optimize option -O1 is specified.

Table 3-6 Jump Instruction Optimization

j instruction	changed machine instruction	
	-O0	-O1
j [cc,]address	jp [cc,]address	jp [cc,]address or jr [cc,]address or jrl [cc,]address

- Subroutine call instruction (cal)

Subroutine call instruction is converted to the smallest machine instruction, considering the range to branch destination, displacement, absolute branch or relative branch, when the optimize option -O1 is specified.

Table 3-7 Subroutine Call Instruction Optimization

j instruction	changed machine instruction	
	-O0	-O1
cal [cc,]address	call [cc,]address	call [cc,]address or calr [cc,]address

3.4.1 Numeric Range

The range of numeric resulting from a constant expression in an operand depends on the mnemonic and addressing mode. A warning message is output and the overflow part is truncated if the value exceeds the anticipated range.

3.5 Assembler Directives

The Assembler directives include a range of instructions such as controlling how the assembler functions. Assembler directives can be classified according to their function, as follows:

Directives controlling modules and sections

module	end	section	align	org
--------	-----	---------	-------	-----

Directives defining data

db	dw	dp	dd	dl
dfb	dfw	dfp	dfd	dfi
dsb	dsw	dsp	dsd	dsl
fdatas	fdatad	fdatdx		

Directives for inter-modular symbol referencing

public	extern
--------	--------

Directives for defining symbols

equ

Directive Notation

Assembler directives do not normally differentiate between uppercase and lowercase letters.

3.5.1 Defining Modules

module

[Description Format]

module <Module_name>

[Function]

Declares the name of a module.

[Explanation]

- The identifier specified in <Module_name> becomes the name of the module.
- This directive is written in the assembly source file header.
- This directive can be specified only once per file. If <Module_name> is specified more than once in a file, a warning is output and the second and subsequent declarations are ignored.

- When this directive is omitted, the module takes the name of the assembly source file excluding the suffix.

end**[Description Format]**

end

[Function]

Declares the end of a module.

[Explanation]

- This directive indicates the end of a module.
- This directive is written in the end of an assembly source file.
- An error is output if this directive is not specified.
- A warning is output if source statements appear after the end directive, and all such source statements are ignored by the assembler. However, they are output to the assembler list file.

3.5.2 Defining Sections

When defining a section according to machine instructions, variables, and constant types, use the section directive. The section directive is valid until next section directive or end directive appears. The section has 3 types as follows.

code section This is a section consolidated machine instructions.

data section These are data that change values during program execution consolidated as memory allocation units.

romdata section These are data that do not change values during program execution consolidated as memory allocation units.

section**[Description Format]**

<Section_name> section <Type> [<Allocation>] [<Alignment>]
--

[Function]

Declares that the program or data that follows this section directive belongs to <Type> section.

[Explanation]

- <Section_name>
Specifies a section name. A section name is character string that can be used as a identifier.
- <Type>
Specifies a section type.
- <Allocation>
The location when allocating a section on memory is specified using displacement or absolute address.
When <Allocation> is omitted, it is regarded as large.

- Displacement specification

Specify the displacement either small or medium or large. When the displacement is specified, the section become a relative address section. However, only data section can specify small.
- Absolute address specification

Specify a constant expression, in the format `abs = <Absolute Address>`. When an absolute address is specified, the section become an absolute address section that allocation address is fixed. The identifier included an absolute address section has absolute address.
- <Alignment>

Specify the alignment when a section is allocated on memory, in the format `align=[<Expression 1>][,<Expression 2>]`.

 - <Expression 1> shows the alignment of the section start.
 - <Expression 2> shows the alignment between each data in a section.

When <Alignment> is omitted, it is regarded as "align=1,1". Although <Expression 1> and <Expression 2> are omissible, respectively, both are not simultaneously omissible.

<Expression 1> specifies the value which can be expressed with power-of-2. When it is not a value which can be expressed with power-of-2, it becomes warning, and it is revalued to the nearest 2^n .

3.5.3 Directives for Inter-modular Identifier Referencing

Identifiers such as variables and labels defined in a section cannot simply be referenced from another module. Identifiers should be declared using one of the following directives to allow them to be referenced from other modules.

public directive The public directive allows identifiers defined within one module to be referenced from other modules.

The identifier declared by this directive is called a public symbol.

extern directive The extern directive allows the referencing of identifiers whose value are defined in other modules and undefined in current module.

The identifier declared by this directive is called an external reference symbol.

public

[Description Format]

<code>public <Symbol>[, <Symbol> , , <Symbol>]</code>

[Function]

Declares <Symbol> as a public symbol.

[Explanation]

- Enables <Symbol> to be referred from other modules.
- A <Symbol> must be an identifier with a defined value in the module.
- This directive can appear anywhere in the assembly source file.

extern

[Description Format]

```
extern    [<Displacement>]    <Symbol>[,<Symbol>,<Symbol>,...,<Symbol>]
```

[Function]

Declares <Symbol> to be an external reference symbol.

[Explanation]

- Declares that the value of <Symbol> defined in another module can be referred.
- When the value of <Symbol> cannot be defined within current module.
- This directive can appear anywhere in the assembly source file.
- In <Displacement> specify the displacement specified in the section directive for the section (in another file) in which the symbol is actually defined.
- When a symbol in an absolute section is declared by "extern", use the extern directive to specify the appropriate displacement in the mapping address in the section directive.
- When the displacement specified in the section directive and the <Displacement> specified in this directive are mismatched, an error occurs at linking.
- When <Displacement> is omitted, it shall be regarded as large.

3.5.4 Data Area Definition Directives

The following four data area definition directives are used to define memory block for data and to initialize the memory block.

d <Size>	: definition directive for memory area of integer data with initial value (1)			
db	dw	dp	dd	dl
df <Size>	: definition directive for memory block of integer data with initial value (2)			
dfb	dfw	dfp	dfd	dfi
ds <Size>	: definition directive for memory block of integer data without initial value			
dsb	dsw	dsp	dsl	
fdata <Size>	: definition directive for memory area of floating-point data with initial value			
fdatas	fdatad	fdatax		

Table 3-8 Data Definition Directives

Size	Definition with initial value	Continuous area definition with initial value	Definition without initial value
8-bit integer area	db	dfb	dsb
16-bit integer area	dw	dfw	dsw
24-bit integer area	dp	dfp	dsp
32-bit integer area	dd, dl	dfd, dfl	dsd, dsl

db, dw, dp, dd, dl

[Description Format]

d<Size> <Expression>[,<Expression>, ,<Expression>]

[Function]

Defines the number of integer data areas specified in <Expression> of the size specified in <Size>. Each defined area is initialized with the value specified in <Expression>.

[Explanation]

- Specify the size of the area to be allocated to one integer by specifying one of the following characters in <Size>.

B	Area for 8-bit integers
W	Area for 16-bit integers
P	Area for 24-bit integers
D or L	Area for 32-bit integers
- <Size> cannot be omitted.
- <Expression> is an expression for initializing the defined area.
- Other than absolute expression can be specified in <Expression>.
- An error occurs when <Expression> does not evaluate to a numerical value.
- When "B" is specified in <Size>, character string can be specify in <Expression>'s place. <Character string> is specified by character string for initializing the defined area. In this case, "\0" is not added to the end of the area. The maximum limit of length of character string by <Character string> is 511 byte. Error is output when the limit exceeds.
- When the value of <Expression> is greater than the size specified in <Size>, a warning is output and the overflow (high bits) is ignored.
- A comma (,) must be inserted between <Expression>s.

dfb, dfw, dfp, dfd, dfi**[Description Format]**

df<Size> <Expression 1>,<Expression 2>

[Function]

Defines the number of contiguous integer data areas specified in <Expression 1> of the size specified in <Size>. All defined areas are initialized with the value specified in <Expression 2>.

[Explanation]

- Specify the size of the area to be allocated to one integer by specifying one of the following characters in <Size>.

B	Area for 8-bit integers
W	Area for 16-bit integers
P	Area for 24-bit integers
D or L	Area for 32-bit integers
- <Size> cannot be omitted.
- <Expression 2> is an expression for initializing the defined area.
- Other than absolute expression can be specified in <Expression 2>.
- An error occurs when <Expression 2> does not evaluate to a numerical value.
- When "B" is specified in <Size>, character string can be specify in <Expression 2>'s place. <Character string> is specified by character string for initializing the defined area. In this case, "\0" is not added to the end of the area. The maximum limit of length of character string by <Character string> is 511 byte. Error is output when the limit exceeds.
- When the value of <Expression 2> is greater than the size specified in <Size>, a warning is output and the overflow (high bits) is ignored.
- <Expression 1> is the number of defined areas and must be an absolute expression. If not, an error occurs.

- If <Expression 1> does not evaluate to a numerical value, an error occurs. Forward reference of an absolute expression is also not allowed.

dsb, dsw, dsp, dsd, dsl**[Description Format]**

ds<Size> <Expression>

[Function]

Defines the number of contiguous integer data areas specified in <Expression> of the size specified in <Size>. The areas are not initialized.

[Explanation]

- Specify the size of the area to be allocated to one integer by specifying one of the following characters in <Size>.

B	Area for 8-bit integers
W	Area for 16-bit integers
P	Area for 24-bit integers
D or L	Area for 32-bit integers
- <Size> cannot be omitted.
- <Expression> is the number of defined areas and must be an absolute expression. If not, an error occurs.
- <Expression> must be a positive integer. A warning is output for values of 0 or less and the default value of 1 is assumed.
- If <Expression> does not evaluate to a numerical value, an error occurs. Forward reference of an absolute expression is also not allowed.

fdatas, fdatad, fdatax**[Description Format]**

fdata<Size> <Expression>[, <Expression> , , <Expression>]

[Function]

Defines only number of <Expression> of floating-point data areas whose size specified in <Size>. All defined areas are initialized with the value specified in <Expression>.

[Explanation]

- Specify the size of the area to be allocated to one floating-point number by specifying one of the following characters in <Size>.

S	Area for 32-bit single-precision floating-point number
D	Area for 64-bit double-precision floating-point number
X	Area for 80-bit extended double-precision floating-point number
- <Size> cannot be omitted.
- <Expression> is an expression for initializing the defined area.
- The evaluation result of <Expression> must be a floating constant.
- When the evaluation result of <Expression> departs from the description rules of the floating constant except for the integers, an error is output. Thus the expression value is regarded as 0.
- When an integer is described in <Expression>, a warning is output. Thus, the integer is converted to a floating constant in accordance with <Size>.

- When the floating-point number size of <Size> differs from <Expression> that, a warning is output. Thus, the <Expression> value is converted to the size specified with <Size>.
- The exponent size of the value specified with <Expression> exceeds the exponent size of the floating-point data specified with <Size>, a warning is output on the exponent to be checked. Thus, the exponent size of the value specified with <Expression> is converted to the maximum exponent can be specified with <Size>.

3.5.5 Equ Directives

In assembly language, identifiers can be defined for the constants used in an assembly source program.

equ

[Description Format]

<Identifier>	equ	<Expression>
--------------	-----	--------------

[Function]

Defines a constant used in all modules as <Identifier>.

[Explanation]

- The value specified in <Expression> is set to <Identifier>.
- <Expression> must be an absolute expression. If not, a warning is output and the value becomes 0.
- The value of a identifier set by this directive cannot be changed.
- An error results if the equ directive is used to redefine a identifier defined by another directive.
- <Identifier> defined by this directive can be declared as a public symbol by a public directive.

3.5.6 Align Directives

align

[Description Format]

align <Expression>

[Function]

Aligns the location counter to a boundary value.

[Explanation]

- If the value of the location counter for the current section is not at the boundary specified in <Expression>, the value of the location counter is adjusted (rounded up) to the boundary value specified in <Expression>.
- Specify in <Expression> an absolute expression that will have been defined when the align directive is encountered.
- An align directive takes precedence over align in the section directive. That is, when "align=<Expression 1>,<Expression 2>" is specified using the section directive, only the next location counter encountered by this directive is adjusted to the value specified in <Expression> in the align directive. The next or later location counter returns to the value as specified in the align of the section directive.

3.5.7 Org Directives

org

[Description Format]

org <Expression>

[Function]

Changes the value of the location counter.

[Explanation]

- Changes the value of the location counter for the current section to the value specified in <Expression>. That is, <Expression> is the address itself in the case of an absolute section, or the offset from the starting address of the section in the case of a relocatable section.
- <Expression> must be an absolute expression defined when this directive is encountered.
- The value specified for the new location counter must be greater than the current location counter.

3.6 Control Instructions

Assembler control instruction is a instruction for controlling operation of assembler. Description form attaches "\$" to a head and describes "\$" and a control instruction continuously. There is no differentiation between uppercase and lowercase letters in an assembler control instruction.

3.6.1 File Include Function

\$include

[Description Format]

\$include "<Filename>"

[Function]

Reads in the assembly source file specified in <Filename>.

[Explanation]

- This instruction reads in and expands the assembly source file specified in <Filename> at the position of the \$include instruction.
- When the files are nested exceeding the assembler maximum limit, an error occurs.
- When no path or a relocatable path is specified in <Filename>, the file is searched in order of the following directories:
 - (1) The directory of the source file during the assembling processing.
 - (2) The directory specified with -I option (when specified more than once, the paths are searched in the order specified).
 - (3) The include directory under the directory specified in the environment variable THOME900.

3.6.2 \$maximum Control Instruction

\$maximum

[Description Format]

\$maximum

[Function]

CPU register mode is set up Maximum mode.

[Explanation]

- This instruction is certainly written at the head of an assembly source file.
- CPU register mode of each TLCS-900 Family CPU supports only Maximum mode. Minimum mode is not supported.

Chapter 4 Assembler List File Format

Reading the machine instructions and corresponding parts source file

Location	Object	Ins	Line	Source Statement
[a]	[o][r]+[n]	[l1]	[l2]	[s]

- [a] This shows the offset value of the program within the assembler source file. After linking, the memory allocation address is determined.
- [o] This shows the machine instructions corresponding to the source program.
- [r] R is displayed when a relocatable value is included.
- [n] Shows the nest level of include.
- [l1] This shows the line number. When an include file is included, this shows the allocated line number for each file.
- [l2] This shows the line number. This is a serial number that is not related to include files, etc.
- [s] This shows the assembler source program.

Symbol Table Format

Symbol	Category	Value	Attribute	Cross_reference
[sym]	[c1][c2][c3]	[v]	[a1][a2][a3]	[cr]

[sym] This shows the symbol name.

[c1] This shows the information type.

D	data
C	code
R	romdata

[c2] This shows the displacement of the data to be allocated.

S	small (tiny)
M	medium (near)
L	large (far)

[c3] This shows the information type.

LAB	Label
VAR	Variable name
NUM	Numerical value
MOD	Module name
SEC	Section name

[v] This shows the offset within an assembler source file at which a symbol is allocated.

[a1] This shows the address determination method.

R	Relocatable
A	Absolute

[a2] This shows the symbol linkage. However, when this is a numerical value, it shows the size.

PUB public

EXT external

Blank local

Numerical value

Section size (When c3 is SEC)

[a3] This shows the section name. However, this is displayed only when the section is relocatable.

[cr] This shows the source line cross reference. <Num># shows the symbol definition line, and others show referenced lines.

Chapter 5 Assembler Options

-I Specify Search Path for Include Files

[Description Format]

-I<Path>

[Function]

Specifies the search path for include files.

[Explanation]

- This option specifies the search path for include files specified in include instruction. <Path> cannot be omitted.
- This option can be specified multiple times. When multiple options specified, the path are searched in the order in which the path are specified.
- Searches are performed according to the path specified in the -I option when the filename is specified with a relative path. When a path is specified with absolute path, the specified file is read in without being searched.
- The file is searched in order of the following directories:
 - (1) The current directory which the source file during the assembling processing.
 - (2) The directory specified with -I option (when specified more than once, the paths are searched in the order specified).
 - (3) The include directory under the directory specified in the environment variable THOME900.

-J Recognize the Kanji Code (Japanese version only)

[Description Format]

-J

[Function]

Recognizes the kanji code.

[Additional Note]

Do not use this option for English version.

-Nb Select CPU Type

[Description Format]

-Nb[<CPU Type>]

[Function]

Specifies CPU type of TLCS-900 Family.

[Additional Note]

- When <CPU Type> is omitted, it is considered that 0 was specified as <CPU Type>.
- When this option is omitted, it is considered that -Nb0 was specified.
- <CPU Type> are as follows:

Table 5-1 CPU Type

CPU Type	Function
0	TLCS-900 series
1	TLCS-900/L series, TLCS-900/L1 series
2	TLCS-900/H series
3	TLCS-900/H1 series

-O Specify Optimization Level**[Description Format]**

```
-O[<Optimization Level>]
```

[Function]

Specifies the optimization level of output code with the Assembler.

[Explanation]

- When only "-O" option is specified without <Optimization Level>, it is recognized as "-O1" option having been specified, and optimization is performed. When you do not carry out the optimization by assembler, it is "-W" option of compiler driver and please pass "-O0" option to assembler.

Table 5-2 Optimization level

Level	Function
0	No optimization with the Assembler
1	Optimization with the Assembler

-V Output Version Number**[Description Format]**

```
-V
```

[Function]

Outputs the Assembler version number to standard output.

[Explanation]

- When the assembler is activated, startup messages such as the assembler version number are output to standard output.
- This option cannot be included in files for which -f option is specified.

-XE Ignore Escape Sequence**[Description Format]**

```
-XE
```

[Function]

Interprets the backslash (\) not as the first symbol in the escape sequence but as a normal character.

[Explanation]

- Use this option when the assembler source file does not contain an escape sequence.

-e Create Error List File**[Description Format]**

-e <Filename>

[Function]

Outputs all error messages to one file as an error list file.

[Explanation]

- This option outputs all errors and warnings that occur during assembler execution to the file specified in <Filename>.
- When a fatal error occurs, processing ends immediately and no error list file can be created.

-f Read Option List File**[Description Format]**

-f<Filename>

[Function]

Reads the options from an option list file containing a startup option list.

[Explanation]

- Describes in advance in a text file the option to be specified and specified the file as <Filename>.
- It is possible to describe options on multiple lines within an option list file.
- This option can be specified multiple times to specify multiple option list files.

-g Create Debugger Information at assemble phase**[Description Format]**

-g[<Debug Level>]

[Function]

Outputs source level debugging information or symbolic debugging information to an object file.

[Explanation]

- Specify the Debug Level as a value of 0 or 1 in <Debug Level>.
- When only "-g" option specify without <Debug Level>, it is recognized as "-g0" option having been specified.
- When this option is omitted, no debugging information is output to the object file.

Table 5-3 Debug Information

Level	Function
0	Outputs source level debugging information.
1	Outputs symbolic debugging informations.

-l Create an Assembler List File**[Description Format]**

-l[<Suboption>]

[Function]

Creates an assembler list file.

[Explanation]

- An assembler list file is generated by the file name which changed the suffix of the source file name to ".lst".
- The information output to the assembler list file is controlled according to <Sub Option>.
- Multiple suboptions can be specified after option -l. However, "f" suboption which needs the argument must be specified last.
- The type of suboption are as follows:

Table 5-4 Suboptions of option -l

Suboptions	Function
No suboption	Outputs a basic format assembler list file.
f<Filename>	Outputs by adding a name <Filename> to the assembler list file. <Filename> cannot be omitted.
x	Outputs cross reference information to an assembler list file.

-o Specify an Output Filename

[Description Format]

```
-o<Filename>
```

[Function]

Specifies the filename of the output file.

[Explanation]

- The output file is created with the name specified in <Filename>.
- The suffix of the filename in <Filename> is not checked.
- When this option is omitted, the output file is generated by the file name which changed the suffix of the source file name to ".rel".
- Assembly is stopped and an error occurs if the same filename is specified for an object file and a source file.

-w Specify Warning Level

[Description Format]

```
-w[<Warning Level>]
```

[Function]

Specifies the warning level.

[Explanation]

- Specify the warning level as a number in <Warning Level>(0 - 1).
- When this option is omitted, it is recognized as "-w1" option having been specified and all warnings are output.
- Specifying "-w" option without <Warning Level> is equivalent to option "-w0", and no warnings are output.

Chapter 6 Assembler Limitation

Table below shows the Limitation of the Assembler.

Table 6-1 Limitation of Assembler

Item	Limitation
Number of lines of a source file	no limitation
Number of characters of one line	no limitation
Number of characters of absolute path	259 characters
Number of identifiers	no limitation
Number of sections	no limitation
Number of characters of identifier	1024 characters
Number of expressions in one directive db/dw/dp/dd/dl	99 times
Number of characters in one directive db/dw/dp/dd/dl	511 characters
Number of expressions in one directive dfb/dfw/dfp/dfd/dfi	no limitation
Number of characters in one directive dfb/dfw/dfp/dfd/dfi	511 characters
Value of expressions in one directive dsb/dsw/dsp/dsd/dsl	no limitation
Number of expressions in one directive fdatas/fdatad/	99 times
Nest level of \$include directive	8 levels
Number of \$include instructions per translation unit	99 times
Number of times that option -I can be specified	31 times

Part 3 The Linker

Chapter 7 The Linker

7.1 Input and Output Files

Input and Output Files

The suffixes used in the names of input and output files of the Linker are as below.

Table 7-1 Input and Output files of the Linker

Suffix	File type	Classification
.rel	Relocatable object files	Input and output
.lib	Library files	Input
.lcf	Link Command files	Input
.abs	Absolute object files	Output
.map	Map files	Output

- Relocatable object files

These object files, which includes relocatable object files, are input and output by the Linker. These files are binary format files that complies with the IEEE695 object module format.

- Library files

The library files consist of multiple relocatable modules collected into one by the librarian. The required modules are extracted from these files during linking.

These files are binary format files that complies with the IEEE695 object module format.

- Link command files

These text files contain entries specifying the linking sequence and allocation of memory. The linker reads the link command file and performs the linking according to its content.

- Absolute object files

The absolute object files are output by the linker. The internal and external symbols in these files have absolute addresses assigned to them. And the files is convertible for the object format which can be used by EPROM writer etc.

These files are binary format files that complies with the IEEE695 object module format.

When the error occurs during link processing, this file is not output.

- Map files

The map files are text files containing information gained from the linking process. They include information about sections after they have been linked, on symbols, and on errors.

When the fatal error occurs during link processing, this file is not output.

Chapter 8 Link Command File

8.1 Basic Link Command File

8.1.1 Reserved Words

Reserved words of the link command file are listed below. They are not available as user-defined symbols. However, no error results and they are processed correctly if they are used as section names in the object file. Reserved words do not differentiate between uppercase and lowercase letters.

addr	align	copy	dsect	len	length	memory
next	noload	org	origin	overlay	sections	sizeof

8.1.2 Identifiers

Identifiers consists of any combination of letters, numbers, period (.), and underscore (_). The first character of an identifier must be except a number. The identifier is guaranteed up to the maximum effective length of character. And the identifiers are classified predefined symbols, and user-defined symbols.

Predefined Symbol

The predefined symbols are used as only memory names. Therefore, they are also called the predefined memory names. Predefined symbols do not differentiate between uppercase and lowercase letters. See Section 8.2.3, "Predefined Memory" for details of the predefined memory.

User-Defined Symbol

Identifiers described by assembler, such as section names and labels, are all user-defined symbols. Also identifiers can be defined in a link command file.

Even if a reserved word of assembly language is included in a link command file, the linker does not output any error. User-defined symbols differentiate between uppercase and lowercase letters.

8.1.3 Expressions

Expressions consist of integer, and identifiers linked by operators.

Integer

The linker handles numeric values as unsigned 32-bit, and ignores the part exceeded. The linker does not handle real numbers. The alphabet used for integer do not differentiate between uppercase and lowercase letters.

Binary:	Expressed as strings of 0s or 1s starting with 0y
Octal:	Expressed as strings of numbers 0 to 7 starting with 0
Decimal:	Expressed as strings of numbers 0 to 9 starting with other than 0
Hexadecimal:	Expressed as alphanumeric strings consisting of numbers 0 to 9, a to f, and starting with 0x

8.1.4 Location Counter

The location counter shows the current address. The value of location counter is shown in the period (.) in the section definition part of a link command file.

The location counter is updated each time the linker allocates a section to a usable memory area.

The value of the location counter can be changed using an assign statement, described later. However, note that the value of the location counter can not be decreased.

8.1.5 Operators and Their Order of Precedence

The operators have the same function as in C language. The available operators and their order of precedence are as follows.

Table 8-1 Operators and Their Order of Precedence

Priority	Operators	
1	Parentheses	(,)
2	Unary operators	! , ~ , -
3	Binary multiplication/division	* , /
4	Binary addition/subtraction	+ , -
5	Shift	< , >
6	Comparison	< , > , <= , >=
7	Comparison	== , !=
8	Bitwise AND	&
9	Bitwise OR	
10	Logical AND	&&
11	Logical OR	
12	Assign operators	= , +=

8.1.6 Functional Operators

The functional operators can be used in a only output specification field of a section definition part, and perform operations on defined sections and integer. See Section 8.3, "Section Definition Part" for details of the section definition part.

addr	align	next	org	sizeof
------	-------	------	-----	--------

addr

[Description Format]

```
addr(<Section_name>)
```

[Explanation]

- Returns the start address of the defined output section specified in <Section_name>.
- An error is occurs if the start address of the specified section cannot be determined.

align

[Description Format]

```
align(<Expression>)
```

[Explanation]

- Return the current location counter which is aligned by the value of <Expression>.
- When this operator is specified in an assign statement, returns the value of the current location counter aligned by the value of <Expression>.
- When this operator is specified as the alignment specification of an output section, this operator specifies the alignment of the output section.

next**[Description Format]**

`next (<Integer>)`

[Explanation]

- The address which a section is not allocated within the defined memory, moreover, multiple of the value specified by <Integer> are returns.

org**[Description Format]**

`org (<Section_name>)`

[Explanation]

- Returns the allocation address of the defined output section specified in <Section_name>.
- An error results if the allocation address of the specified section cannot be determined. The address of the specified section is required to be previously specified with the allocation address specification (`org=<Expression>`) in the section definition part. See Section 8.3.4, "Output Specification Field".

sizeof**[Description Format]**

`sizeof (<Section_name>)`

[Explanation]

- Returns the size in bytes of the defined output section specified in <Section_name>.

8.1.7 Assign Statements

Assign statements are used when defining or re-defining public symbols.

[Description Format]

`<User-defined Symbol> = <Expression>;`
`<User-defined Symbol> += <Expression>;`

[Explanation]

- The semicolon (;) must terminate an assign statement.
- Assign statements can be described anywhere in the link command file.
- When allocating sections, the linker evaluates the expressions in the order in which they are written and assigns the specified symbols to them.
- The value of the location counter can be adjusted by using assign statements in the section definition part. However, the value of the location counter cannot be decreased.

8.1.8 Comments

Comments can be written anywhere in the link command file. Comments are enclosed in `/*` and `*/`. Comments can also span two or more lines. Comments cannot be nested.

8.2 Memory Definition Part

8.2.1 Function of Memory Definition Part

By defining the address, size, and attributes at the memory definition part, the address space where sections are allocated is defined. Sections cannot be allocated to other than the address space defined here.

When there is no memory definition in a link command file, the linker regards this as all memory spaces being valid, and as having all attributes.

8.2.2 Memory Definition Part Format

[Description Format]

```
memory {
    <Memory_name> [(<Attribute>)] : org=<Address> [,] len=<Size>
    <Memory_name> [(<Attribute>)] : org=<Address> [,] len=<Size>
    ...
}
```

[Explanation]

<Memory_name>

This is the name given to the defined memory area. The memory name can be a predefined or user-defined memory name. User-defined memory names must not exceed the maximum effective length of character. Memory names are used only by the linker, and are therefore not passed to the output file.

<Attribute> Specify the attribute of the memory area to define.

R	Readable
W	Writable
X	Executable
I	Initializable

<Attribute> differentiate between uppercase and lowercase letters.

If specification is omitted, it will be regarded that specify all the attributes of "R", "W", "X", and "I".

The linker initializes any empty area when it exists in a section allocated to a memory area for which the "I" attribute is specified. See section 8.3.7 "Padding Field" for details of initialization.

Table below shows the relationship between the types of sections specified in the assembly language section directive and the memory attribute specified in the memory definition part.

Table 8-2 Section Type and Required Memory

Section type	Required memory attribute
data	RW
romdata	R
code	RX

The linker checks the attributes when allocating a section to a memory space. If not corresponding memory attribute has been specified, an error occurs. For example, the "R" (readable) and "X" (executable) attributes are required for memory to which a CODE section is allocated.

- <Address>** Specify the start address of the memory area to define with 32-bit unsigned integer. You can also use "origin" in place of "org" to specify "origin=<Address>".
- <Size>** Specify the size in bytes of the memory area to define. You can also use "length" in place of "len" to specify "length=<Size>". Delimit <Address> and <Size> with a comma or space.

8.2.3 Predefined Memory

If predefined memory name used in memory name, sections are allocated automatically to memory area corresponding to section directive in an assembler source file unless otherwise specified in the section definition part. Predefined memory have the following meanings.

code.m, code.l

A code area is an area to which machine instructions are allocated. Sections specified with the "code" type in the assembly language section directive are assigned to this memory area.

data.s, data.m, data.l

A data area is an area to which data with values that change during program execution are allocated. A data area has the "RW" attributes. Sections specified with the "data" type in the assembly language section directive are assigned to this memory area.

romdata.m, romdata.l

A romdata area is an area to which data (constants) with values that do not change during program execution are allocated. A romdata area has the "R" attribute. Sections specified with the "romdata" type in the assembly language section directive are assigned to this memory area.

The predefined memory "data.s" corresponds to displacement "small" of the assembly language section directive. Similarly, the "code.m", "data.m", and "romdata.m" corresponds to "medium", and the "code.l", "data.l", and "romdata.l" corresponds to "large". The section type specified in the assembly source program must match the attribute of the predefined memory to which it is assigned. For relocatable sections, both section displacement and type must match. For absolute sections, the section type must match.

Table 8-3 Section type and Predefined memory name

Section type	Displacement	Predefined memory name
data	small	data.s
	medium	data.m
	large	data.l
romdata	medium	romdata.m
	large	romdata.l
code	medium	code.m
	large	code.l

Displacement of Predefined Memory

The predefined memory can be defined in duplicate address, when the memory are same types and different displacement. For example, section "data small" allocated to the area "data.s" defined in the memory definition part, then section "data medium" can be allocated continuously after "data small" if any area remains in "data.s". Also, user-defined memory can not be defined in duplicate address.

8.3 Section Definition Part

8.3.1 Function of Section Definition Part

The section definition part specifies the linking order of input sections and the allocation of them to memory.

When the section definition part is omitted, the sections are linked in the input order, and are allocated. At this time, sections with the same name are linked to contiguous memory. However, sections with the same name must have the same attributes in all input files.

8.3.2 Section Definition Part Format

[Description Format]

```
sections {
    <Output Section Name> [<Output Specification>] [<Attribute>] :
        {<Input Section Specification>}
        [=<Padding>] [> <Output Memory>]

    <Output Section Name> [<Output Specification>] [<Attribute>] :
        {<Input Section Specification>}
        [=<Padding>] [> <Output Memory>]

    ...
}
```

[Explanation]

<Output Section Name>

Specify the output section name.

<Output Specification>

Specify the allocation address, alignment, and size of the output section.

<Attribute> Applies a special attribute to the output section. This specification can be omitted.

<Input Section Specification>

Specify the input section name.

<Padding> Specify the numerical value to initialize padding.

<Output Memory>

Specify to allocate the output section to the user-defined memory area defined in the memory definition.

8.3.3 Output Section Name Field

The section name for the output section is specified in the output section name field. The output section name can specify the same name as the input section name.

8.3.4 Output Specification Field

The allocation address, alignment, size, and start address of the output section are specified in the output specification field.

[Description Format]

[<Allocation Address>][<Alignment>][<Size>][<Start Address>]
--

[Explanation]

<Allocation Address>

Specify the address in memory to which the output section will be allocated.

<Alignment>

Specify the alignment for adjusting the allocation of the output section.

<Size>

Specify the minimum size of the output section.

<Start Address>

The start address is the reference address to be used when applying an absolute address to a symbol. The execution address is specified as the start address when the area storing objects is different from the area used at the execution. For example, when the variables with initial values used, initial value allocate to ROM and variables allocate to RAM. In this case, the address allocated body of variables will be the start address.

- All specifications are optional. You can also specify the memory name to be applied without specifying the allocation address. See 8.3.8 "Output Memory Field" for details of specifying the output memory.

Allocation Address Specification

org

[Description Format]

org = <Expression>

[Function]

Specify the memory allocation address of the output section.

[Explanation]

- When this directive is omitted, the allocation address and the start address are the same.
- The input section types (code, data, and romdata) and the type of memory of the memory definition which the allocation address corresponds do not have to match.
- The allocation address and alignment cannot be specified at once.

Alignment Specification

align

[Description Format]

align = <Expression>

[Function]

Specify the memory allocation address of the output section, the address is multiple of the value specified in <Expression>

[Explanation]

- <Expression> specifies the value which can be expressed with power-of-2.
- The allocation address and alignment cannot be specified at once.

Size Specification**len****[Description Format]**

```
len = <Expression>
```

[Function]

Specify the size of the output section.

[Explanation]

- When the value of <Expression> is greater than the total size of the input sections, the size of the output section is the value specified in <Expression>.

Start address Specification**addr****[Description Format]**

```
addr = <Expression>
```

[Function]

Specify the start address of the output section.

[Explanation]

- Specify the start address when the allocation address and the start address of the output section are not the same.
- When this directive is omitted, the allocation address and the start address are the same.
- The start address must be within the area defined in the memory definition.
- Specification of the start address is described after the specification of the allocation address.

8.3.5 Attribute Field

The attribute field is specified when a special attribute is applied to the output section. There are four attributes: DSECT, COPY, NOLOAD, and OVERLAY. Table below shows the differences between sections with and without these attributes.

Normally, these attributes do not need to be applied to an output section.

Table 8-4 Attribute Field

Attribute	Object data	Overlap
None (normal)	Output	No
DSECT	Not output	Yes
COPY	Output	Yes
NOLOAD	Not output	No
OVERLAY	Output	Yes

DSECT**[Description Format]**

(DSECT)

[Explanation]

Specify the dummy section attribute (no body) for the output section. Because, practically, no actual memory is allocated to a section with the DSECT attribute, it can overlap with other sections.

Also, dummy sections can be allocated to memory areas not declared in the memory definition.

COPY**[Description Format]**

(COPY)

[Explanation]

The COPY attribute is identical to the DSECT attribute except that object data is output.

NOLOAD**[Description Format]**

(NOLOAD)

[Explanation]

sections with the NOLOAD attribute are identical to normal sections except that no object data is output.

OVERLAY**[Description Format]**

(OVERLAY)

[Explanation]

- Sections with the OVERLAY attribute are allocated to memory in the same way as normal sections, and object data is output.
- OVERLAY sections differ from normal sections in that these can overlap other sections. Normal sections cannot be allocated to an area to which another section has already been allocated. OVERLAY sections, however, can be.
- The allocation address must be specify for OVERLAY sections.

8.3.6 Input Section Specification Field

In the input section specification field, specify file names and input section names included in the file. With the linker, input sections in the specified file are linked. Note that the input sections which are not specified in the input section specification field, the output sections will automatically be created with the same name as the input sections. Thus, if two or more input sections have the same name, they will be linked into one in the order in which they appear.

[Description Format]

```
{ <Filename>( <Section_name>... )
  :
  <Filename>( <Section_name>... )}
```

[Explanation]

- Specify a relocatable object file containing the sections to be linked in <Filename>.
- An asterisk (*) can be specified in place of <Filename>, in which case all relocatable object files and library files as target for linking are processed.
- Specify section names to be linked in <Section_name>. The specified section types (code, data, romdata) of the input sections must match. The output section has the same types as the input sections provided special attributes such as DSECT are not specified.
- Multiple sections in the same file can specify in <Section_name>. When specifying multiple sections, delimit the section names with spaces.
- When <Section_name> is omitted, all sections in the specified <Filename> are linked.
- An error is also output and the specification is ignored, when an absolute section is specified as an input section in the section definition part.

8.3.7 Padding Field

The empty areas between input sections that arise when the input sections are linked is called padding. The linker can initialize the padding with the specified value.

[Description Format]

```
=<Numerical Value>
```

[Explanation]

- Specify a value for initializing padding in <Numerical value>.
- Specify a 2-byte numerical value in <Numerical Value>.
- The area is initialized in the order high byte, low byte from the start address for padding regardless of whether the start address is an odd or even. That is, when the padding area is an odd byte, the last byte is initialized as the high byte.
- When the padding specification is omitted, the padding will be initialized with the value specified by -F option when the linker is activated. When the -F option is also omitted, initialization is done with 0.
- <Numerical Value> can be specified to only memory areas with the "I" attribute. An error results if <Numerical Value> is specified for a memory area without the "I" attribute. In this case, the <Numerical Value> specification is ignored.

8.3.8 Output Memory Field

The output memory field is specified so that the output section is allocated to memory defined in the memory definition part. At this time, the RWXI attributes of the memory area are checked against those of the section. If they conflict, an error occurs.

[Description Format]

`> <Memory_name>`

[Explanation]

- The output section is allocated to the memory area specified in <Memory_name>.
- <Memory_name> is the name of a memory area defined in the memory definition part.
- The linker performs allocation to a predefined memory automatically according to the section types, so it is not necessary to specify predefined memory names in <Memory_name>.
- This specification is ignored, when you specify with the allocation address specification(`org`) or the start address specification(`addr`) in output specification field.

8.4 Symbol Definition Part

The link command file can define the symbols. The identifier referred to by ROM/RAM transfer in a startup file is defined here.

8.5 Incremental Linking

The linker can leave the output files as relocatable state. The object file resulted of linking can become an input file for linking target again, thus, linking can be performed incrementally. This function is called the incremental linking.

8.5.1 Processing in Incremental Linking

In incremental linking, the input sections are linked to create an output section, and the result of resolving identifier address is output to a relocatable object file. Searching library, allocation to the memory area, and reallocation of relative expression is not performed.

Unlike in the case of normal linking, it is necessary to specify following using incremental linking.

- Specify option `-r`.
- It is necessary to specify output file name with option `-o`.
- The memory definition part described in a link command file is ignored. Allocation to the memory area is not performed in incremental linking.
- Specify only about section linking in the section definition part. When the memory allocation of the output section is specified, an error is output.
- Assignment statement is not available.

Example of Incremental Linking

[Command Line]

`tulink sample1.lcf testa.rel testb.rel testc.rel`

[Link Command File : sample1.lcf]

```
MEMORY {
    data.m : org=0x1000 len=0x2000
    code.m : org=0x8000 len=0x7000
}
SECTIONS {
    OUT_CODE : { *(code_sec) }
```



```
OUT_DATA : { *(data_sec) }
}
```

Figure 8-1 Sample of Normal Link Command File

To use two-step incremental linking to create the identical file which created using the above link command file (Figure 8-1), the link command files describe as follow.

[Command Line]

```
tulink -r -o testrel.rel sample2-1.lcf testa.rel testb.rel
```

[Link Command File : sample2-1.lcf]

```
/* Cannot specify the memory definition part */
SECTIONS {
    /* Specifies only linking section */
    R_CODE : { *(code_sec) } /* Link code section */
    R_DATA : { *(data_sec) } /* Link data section */
}
```

Figure 8-2 Using link command file for 1st linking

The linker links "code_sec" sections in the relocatable object files "testa.rel" and "testb.rel" to create section "R_CODE", links "data_sec" sections to create section "R_DATA", outputs them to the file "testrel.rel".

[Command Line]

```
tulink sample2-2.lcf testrel.rel testc.rel
```

[Link Command File : sample2-2.lcf]

```
MEMORY { /* Specify the memory definition part */
    data : org=0x1000 len = 0x2000
    code : org=0x8000 len = 0x7000
}
SECTIONS { /* Section definition part */
    OUT_CODE : { *(R_CODE) *(code_sec) }
               /* Link code sections */
    OUT_DATA : { *(R_DATA) *(data_sec) }
               /* Link data sections */
}
```

Figure 8-3 Using link command file for 2nd linking

testrel.rel which is created first and testc.rel which created next, are linked, create an absolute object file. The section is connected R_CODE and code_sec, and R_DATA and data_sec too.

The creating file using two-step incremental linking (Figure 8-2,8-3) is almost the same as the file using one-step linking (Figure 8-1).

Chapter 9 Linker Map File Format

Command file

The contents of Link Command Files are output as is.

If error occurs, the error message is output after the contents of Link Command File.

Input files

This displays the input file names and the module names to be linked. Also displayed here is which libraries are linked.

Link map

Displays an image of a program's allocation on actual memory. Program allocation is specified by a Link Command File.

Memory	This shows the memory name defined by the memory definition part of a Link Command File.
Out-sec	This shows the output section name defined by the section definition part of a Link Command File.
Attri	One of CODE, DATA, or ROMDATA is displayed, and these respectively show code section, data section, and romdata section.
Base	This shows the start address of an output section.
Length	This shows the output section size.
In-sec(In-file)	This shows the input section name, and shows the name of the file in which the input section exists.

Information

This show the following information.

NORMAL	normal section
NORMAL:A	absolute address section
Gap	empty area
DUMMY	DUMMY section
NOLOAD	NOLOAD section
COPY	COPY section
OVERLAY	OVERLAY section

Multiply defined symbols

This displays the multiply defined external definition symbol name and reference file name.

Unresolved external symbols

This displays the unresolved external reference symbol name and reference file name.

Symbol table for <Output File Name>

This displays information related to identifier names that exist in an absolute object file.

Symbol	This shows the identifier name.
Address	This shows the address at which the identifier is allocated.
In-sec	This shows the name of the input section in which the identifier exists.
Cross-reference	This shows which identifiers are cross referenced with what modules

Chapter 10 Linker Options

-F Specify Fill Value for Empty Area in Output Section

[Description Format]

-F<Value>

[Function]

Fills the empty areas in an output section with the specified value.

[Explanation]

- <Value> is specified as a 2-byte numerical value.
- The empty area is filled in the order high byte, low byte of the specified <Value> from the padding start address.
- When the empty area is an odd byte, the last byte is initialized with the high byte of <Value>.
- This option becomes effective is only the memory area which specified 'T' attribute in the memory definition of the link command file. When the memory attribute specification is omitted, the memory area have 'T' attribute by default.

-L Specify Search Path for Input Files

[Description Format]

-L<Path>

[Function]

Specifies the search path for input files.

[Explanation]

- The Linker searches for object files, library files and link command files according to the <Path> specified by this option.
- <Path> cannot be omitted.
- This option can be specified multiple times. When specified more than once, the paths are searched in the order in which the paths are specified.
- When the relative path is specified as <Path>, it is searched as a relative path from the directory which executed the linker.
- The file is searched in order of the following directories:
 - (1) The current directory.
 - (2) The directory specified with option -L (when specified more than once, the paths are searched in the order specified).
 - (3) The lib directory under the directory specified in the environment variable THOME900.

-V Output Version Number

[Description Format]

-V

[Function]

Outputs the Linker version number to standard output.

[Explanation]

- When the linker is activated, startup messages such as the linker version number are output to standard output.

-e Create Error List File**[Description Format]**

```
-e<Filename>
```

[Function]

Outputs all error messages to one file as a error list file.

[Explanation]

- This option outputs all errors and warnings that occur during linker execution to the file specified in <Filename>.
- When a fatal error occurs, processing ends immediately and no error list file can be created.

-g Create Debugger Information at link phase**[Description Format]**

```
-g<Sub Option>
```

[Function]

Outputs debugging information to an absolute object file.

[Explanation]

- This option outputs debugging information in the relocatable object file to an absolute object file.
- When this option is omitted, debugging information is not output to an absolute object file.
- When option -g is specified with the driver, this option is transferred to the linker automatically as option -ga.

Table 10-1 Sub Option for option -g

Sub Option	Function
a	Outputs debugging information of all input files specified as the linker.
m <filename>	Outputs debugging information of the input file specified <filename>. This option can specify multipule. <filename> can not be omitted.

-l Create a Link Map File**[Description Format]**

```
-l<Suboption>
```

[Function]

Creates a map file.

[Explanation]

- The map file is generated by the file name which changed the suffix of the first specified source file name to ".map".
- The information output to the link map file is controlled according to <Sub Option>.
- Multiple suboptions can be specified after option -l. However, 'f' suboption which needs the argument must be specified last.

- The type of suboption are as follows:

Table 10-2 Suboptions of option -l

Suboptions	Function
No suboption	Outputs a basic format map file.
a	Outputs the link information of external variable and local variables to a map file.
f<Filename>	Output a map file which is named <Filename>.
s	Output the link information of static identifier to a map file. -g option is required at the time of compile.
x	Output cross-reference information to a map file.

-o Specify an Output Filename**[Description Format]**

-o<Filename>

[Function]

Specifies the filename of the output file.

[Explanation]

- The output file is created with the name specified in <Filename>.
- The suffix of the filename in <Filename> is not checked.
- When this option is omitted, the output file is generated by the file name which changed the suffix of the first specified input relocatable object file name to ".abs".
- This option must be specified when creating a relocatable object file by specifying option -r.

-r Perform Incremental Linking**[Description Format]**

-r

[Function]

A relocatable object file is created by incremental linking.

[Explanation]

- This option is used when creating a new relocatable object file by linking a relocatable object file with another relocatable object file.
- When this option is specified, also specify the name of the output file using the option -o. An error occurs if the output filename is not specified.
- The input sections are linked according to the section definition part in the link command file. The mapping address is not, however, defined.
- When this option is specified, a warning message is output and the memory definition part is ignored if one exists in the link command file.
- The library files are not linked when this option is specified by the driver.

-u Link Undefined Symbol**[Description Format]**

-u<Symbol>

[Function]

Forcibly links a specified <Symbol>.

[Explanation]

- This option is used to force a module link. It is effective when linking a module from a library file, for example.
- This option can be specified multiple times.
- Do not put in a blank between -u and <Symbol>.

-w**Specify Warning Level****[Description Format]**

-w[<Warning Level>]

[Function]

Specifies the warning level.

[Explanation]

- Specify the warning level as a number in <Warning Level>(0 - 1).
- When this option is omitted, this is recognized as option -w1 having been specified and all warnings are output.
- Specifying option -w without <Warning Level> is equivalent to option -w0, and no warnings are output.

Chapter 11 Linker Limitation

Table below shows the Limitation of the Linker.

Table 11-1 Limitation of Linker

Item	Limitation
Number of characters of section name	1024 characters
Number of characters of memory name	1024 characters
Number of characters of identifier	1024 characters
Number of times that option -L can be specified	8 times

Part 4 The Macro Preprocessor

Chapter 12 The Macro Preprocessor

12.1 Macro Preprocessor Overview

Main function of the Macro Preprocessor is to substitute character strings. For this function, Macro Preprocessor provides a macroprocess function and preprocess function. A processing target of Macro Preprocessor is a source file which is described using Macro Preprocessor language. A part except it merely is just copied to an output file.

Firstly, Macro Preprocessor processes preprocessing directive. After that, it processes macroprocessing directive.

12.2 Input and Output Files

- Table below shows the suffixes used in the names of input and output files of Macro Preprocessor.

Table 12-1 Input and Output files of Macro Preprocessor

Suffix	File type	Classification
.mac	Macro preprocessor source file	Input
.asm	Assembly source file	Output
.med	Macro preprocessor list file	Output

- Macro preprocessor source file
This is an assembly source file that is described using Macro preprocessor language.
- Assembly source file
This is an assembly source file which processing result of Macro Processor.
- Macro preprocessor list file
This is a list file which contains the assembly source file as result of Macro preprocessor, description of each function for Macro preprocessor and expansion form, symbol list and cross references.

Chapter 13 Macro Preprocessor Grammar

13.1 Character Set

The following list shows the characters that can be used when writing a macro preprocessor source file.

Character set:

English alphabets | decimal digits | graphic characters | new-line space character

Graphic characters:

! " # \$ % & ' () * + , - . / : ; < = > ? [\] ^ _ { | } ~ @

In addition to the characters listed above, other characters can be used in comments and text. However, Macro Preprocessor ignores the escape sequence.

13.2 Character String

Macro Preprocessor recognizes following as character string.

- ☐ Character sequence which is put in double quotation.
- ☐ Character sequence which is put in parentheses.

When no character which is put in double quotation or parentheses, Macro preprocessor recognizes empty character string.

13.3 Identifier

Identifier consists of any combination of alphabets (A-Z a-z), numbers (0-9), and underscore(_). And the identifier is classified reserved words, defined macros, and user-defined symbols.

- The first character of an identifier must be an alphabet or underscore.
- The identifier is guaranteed up to the maximum effective length of character. If it exceeds the maximum effective length of characters, exceeded part is truncated, and warning message is output.
- The Macro Preprocessor distinguishes between the uppercase and lowercase letters which are used for a identifier. However, it does not distinguish neither the uppercase nor lowercase letters for the reserved words.

Reserved Word

Reserved words get ready for preprocess function and macroprocess function.

Preprocess function

define	else	elif	endif	error	
if	ifdef	ifndef	include	line	undef

Macro function

assign	chop	define	delayslot	else
elseif	endif	endm	endr	endres
endw	eqs	eval	exit	ges
gts	hi	if	include	labels
len	les	lo	local	lts
macro	membase	memoffset	msg_stderr	msg_stdout
nes	nextreg	num	operator_sign	
regtype	repeat	restrict_macro	set	substr
trigger	typeof	variable	while	

Defined Macro

The following table lists the defined macro.

__TRUE__	__FALSE__	__900__	__TOSHIBA__
----------	-----------	---------	-------------

- __TRUE__ : It is used with the return value of control macro. As for this macro, the value of 1 is defined.
- __FALSE__ : It is used with the return value of control macro. As for this macro, the value of 0 is defined.
- __900__ : The value of 1 is defined when the target MCU is TLCS-900.
- __TOSHIBA__ : The value of 1 is defined, when the target MCU is TOSHIBA MCU. This macor is not defined when conditions are not satisfied.

User-Defined Symbol

User-defined symbol is a identifier which is defined by user.

13.4 Constants

Macro Preprocessor recognizes two types of constant.

Numeric Constants

Macro Preprocessor recognizes integer number as 64 bit length, and calculates the number as signed integer number internally. However, the integer number is output to file by masking to 32 bit length. If calculated result exceeds 32 bit length, Macro Preprocessor outputs a warning message and truncates exceeded part. The alphabet used for integer do not differentiate between uppercase and lowercase letters.

Binary:	Expressed as strings of 0s or 1s starting with 0b or 0y
Octal:	Expressed as strings of numbers 0 to 7 starting with 0
Decimal:	Expressed as strings of numbers 0 to 9 starting with other than 0
Hexadecimal:	Expressed as alphanumeric strings consisting of numbers 0 to 9, and a to f and starting with 0x

Character Constants

Character constants are expressed with enclosing in single-quotes ('). Macro Preprocessor recognizes 4 bytes as character constants.

13.5 Expressions

Macro Preprocessor calculates expression as signed 64 bit length internally. The integer number is output to file by masking to 32 bit length. If calculated result exceeds 32 bit length, Macro Preprocessor outputs a warning message and truncates.

Operator Precedence

The operators have the order of precedence in using. The tables below shows the operator precedence.

Table 13-1 Operator Precedence

Precedence	Operator
1	Unary Operators (~, -, +, !)
2	Multiply, Divide and Modulo (*, /, %)
3	Add and Subtract (+, -)
4	Shift Operation (>>, <<)
5	Comparison (>, <, >=, <=)
6	Comparison (==, !=)
7	Bitwise And (&)
8	Bitwise Or ()
9	Bitwise Exclusive Or (^)
10	Logical And (&&)
11	Logical Or ()

13.6 Trigger Character

The symbol that will instruct the Macro Preprocessor to process a macro is a trigger character. The trigger character is a question mark '?'.

13.7 Comments

The comment starts with a semicolon (;) or two consecutive slashes (//) and extends to the end of the line. When block comment is, enclose multiple lines by "/*" and "*/".

Only the line comment expressed with a semicolon is output to the output file.

13.8 Line Splice

When a new line character immediately follows the backslash (\), the backslash and the new line character has to be ignored and treat the next line as part of the previous line. It is used to divide a long line.

Chapter 14 Preprocess Functions

The Macro Preprocessor can use preprocessing directives same as C language. The directives are as follows.

<code>#if</code>	<code>#ifdef</code>	<code>#ifndef</code>	<code>#elif</code>	<code>#else</code>	<code>#endif</code>
<code>#define</code>	<code>#undef</code>				
<code>#include</code>					
<code>#line</code>	<code>#error</code>	<code>#</code>	<code>##</code>		

Chapter 15 Macroprocess Functions

15.1 Operators

The table below shows the operators of each function.

Conditional Macro

These are macros which processes according to conditions.

Table 15-1 Conditional Macros

Operator	Function
exit	Break expanding according to condition
if	Process according to condition
repeat	Repeat according to specified number of times
while	Repeat according to condition

Replacement Macro

These are macros for character string definition.

Table 15-2 Replacement Macros

Operator	Function
macro	Define replacement string
restrict_macro	Suppress replacement

Numerical Macro

These are macros which controlled numeric number.

Table 15-3 Numerical Macros

Operator	Function
eval	Evaluate expression
variable	Assign value

String Control Macro

These are macros which controlled string.

Table 15-4 String Control Macros

Operator	Function
assign	Divide string
chop	Delete white space
eqs	String1 == String2
ges	String1 >= String2
gts	String1 > String2
len	Return string length
les	String1 <= String2
lts	String1 < String2
nes	String1 != String2
substr	Extract substring from character string

Particular Macro

Table 15-5 Particular Macro

Operator	Function
trigger	Change trigger character

15.2 Conditional Macro

exit

[Description Format]

```
?exit [( <Expression> )]
```

[Function]

Break expanding macros.

[Explanation]

- In case of evaluating result of <Expression> is true, abort text expanding process by ?repeat, ?while, and macro expanding process by ?macro.
- In case of omitting <Expression>, it means true.

[Example]

```
?macro macname arg1
    Macro_body1
    Macro_body2
    ?exit          // abort expanding
    Macro_body3    // do not expand
    Macro_body4    // do not expand
?endm

macname 1          // call

==== Result ====
    Macro_body1
    Macro_body2
```

if

[Description Format]

```
?if ( <Expression> )
    <Text 1>
[?elseif ( <Expression> )
    <Text 2>]
[?else
    <Text n>]
?endif
```

[Function]

Perform condition judgment processing.

[Explanation]

- In case of evaluating result of <Expression> is true, expanding <Text n>.
- In case of evaluating result of <Expression> is false and exists ?else statement, <Text n> of ?else is expanded.
- This function can be nested to 32 level with other function.

[Example]

```
?variable var1, 1

?if(?var1 == 1)
    Process_A
?elseif(?var1 == 2)
    Process_B
?else
    Error_pattern
?endif

==== Result ====
    Process_A
```

repeat

[Description Format]

```
?repeat (<Expression>)
    <Text>
?endr
```

[Function]

Repeat expanding <Text> according to <Expression>.

[Explanation]

- Repeat expanding <Text> for <Expression> times.
- A negative number cannot be specified as <Expression>.
- This function can be nested to 32 level with other function.

[Example]

```
?repeat(3)
    Repeat_body
?endr

==== Result ====
    Repeat_body
    Repeat_body
    Repeat_body
```

while

[Description Format]

```
?while (<Expression>)
    <Text>
?endrw
```

[Function]

Repeat expanding <Text> according to condition.

[Explanation]

- Repeat expanding <Text>, while <Expression> is true.
- This function can be nested to 32 level with other function.

[Example]

```
?variable var1, 3
```

```

?while(?var1 > 0)
    Repeat_body
    ?variable var1, ?var1 - 1    // decrement
?endw

==== Result ====
Repeat_body
Repeat_body
Repeat_body

```

15.3 Replacement Macro

macro

[Description Format]

■ Definition

```

?macro <Identifier> [<Parameter> [, <Parameter> [<Parameter List>]]]
[labels <Label> [, <Label List>]]
    <Text>
?endm

```

■ Call

```
<Identifier> [<Argument List>]
```

[Function]

Define <Identifier> for expanding <Text>.

[Explanation]

- <Identifier> which defined by ?macro can not redefine using ?macro.
- ?macro can redefine <Identifier> which is declared using ?variable, ?assign.
- <Parameter> is described parameter using in <Text>. <Parameter> can be specified 127 times. Separate multiple <Parameter> by comma.
- <Label> is described labels using in <Text>. <Label> can be specified 64 times. Separate multiple <Label> by comma.
- Unique number is added to <label>, when <Text> is expanded. Added number is decimal number from 000 to 999.
- Separate multiple <Argument List> by comma.
- In ?restrict_macro range, expanding of <identifier> is suppressed.
- Recursive pattern is not expanded.
- Function macro cannot be describe as nest definition, when macro define.

[Example]

```

?macro macname arg1, arg2
labels lab1, lab2
    Macro_body
    ?arg1, ?arg2
    ?lab1
    ?lab2
?endm

macname 1, 3    // call
macname 2, 4    // call

==== Result ====

```

```
Macro_body
1, 3
lab1000
lab2000
Macro_body
2, 4
lab1001
lab2001
```

restrict_macro**[Description Format]**

```
?restrict_macro
    <Text>
?endres
```

[Function]

Suppress expanding symbol which is defined by ?macro.

[Example]

```
?macro macname arg1
    Macro_body ?arg1
?endm

?restrict_macro
macname 1    // Suppress expanding
macname 2    // Suppress expanding
?endres
macname 3    // Expanding

==== Result ====
macname 1
macname 2

Macro_body 3
```

15.4 Numerical Macro

eval**[Description Format]**

```
?eval (<Expression>)
```

[Function]

Evaluate <Expression>.

[Explanation]

- Return the result as decimal number.

[Example]

```
?eval(1 + 2 * 3 & 4)    // 4
```

variable**[Description Format]**

- Definition

```
?variable <Identifier>, <Expression>
```

- Call

```
?<Identifier>
```

[Function]

Assign evaluated value of <Expression> to <Identifier>.

[Explanation]

- <Identifier> defined by ?variable can redefine by ?variable.
- Return the result as decimal number.

[Example]

```
?variable a, 1 - 1 && 1
?a          // 0
```

15.5 String Control Macro

assign**[Description Format]**

```
?assign (<Identifier 1>,<Identifier 2>) ([<Character String>])
```

[Function]

Assign <Character String> to <Identifier 1> and <Identifier 2>.

[Explanation]

- <Character String> from beginning to first comma is assigned to <identifier 1>. Remaining <Character String> is assigned to <identifier 2>.
- If comma doesn't exist in <Character String>, <Character String> is assigned to <identifier 1>.

[Example]

```
?assign(str1, str2)( AAA, BBB )
?str1    // expand to " AAA"
?str2    // expand to " BBB "

?assign(str1, str2)(" AAA, BBB ")
?str1    // expand to " AAA"
?str2    // expand to " BBB "
```

chop**[Description Format]**

```
?chop ([<Character String>])
```

[Function]

Delete white spaces in head and tail of <Character String>.

[Explanation]

- In case of specifying particular character (exp parenthesis), use double quotation type in <Character String>.

[Example]

```
?chop( AAA BBB CCC )           // AAA BBB CCC
?chop( " ( AAA BBB CCC ) " ) // ( AAA BBB CCC )
```

eqs**[Description Format]**

```
?eqs ([<String 1>],[<String 2>])
```

[Function]

Compare <String 1> and <String 2>.

[Explanation]

- If <String 1> is equal to <String 2>, return `__TRUE__`. Otherwise, return `__FALSE__`.
- In case of specifying particular character (exp parenthesis), use double quotation type in <String 1> or <String 2>.

[Example]

```
?eqs(AAA,AAA)           // __TRUE__
?eqs(AAA, AAA)          // __FALSE__

?eqs("AAA", "AAA")      // __TRUE__
?eqs("AAA", "AAA")      // __TRUE__
```

ges**[Description Format]**

```
?ges ([<String 1>],[<String 2>])
```

[Function]

Compare <String 1> and <String 2>.

[Explanation]

- If <String 1> is more than or equal to <String 2>, return `__TRUE__`. Otherwise, return `__FALSE__`.
- In case of specifying particular character (exp parenthesis), use double quotation type <String 1> or <String 2>.

[Example]

```
?ges(AAA,AAA)           // __TRUE__
?ges(AAA, AAA)          // __TRUE__
?ges( AAA,AAA)          // __FALSE__

?ges("AAA", "AAA")      // __TRUE__
?ges("AAA", " AAA")     // __TRUE__
?ges(" AAA", "AAA")     // __FALSE__
```

gts**[Description Format]**

```
?gts ([<String 1>],[<String 2>])
```

[Function]

Compare <String 1> and <String 2>.

[Explanation]

- If <String 1> is more than <String 2>, return `__TRUE__`. Otherwise, return `__FALSE__`.
- In case of specifying particular character (exp parenthesis), use double quotation type <String 1> or <String 2>.

[Example]

```
?gts(AAA,AAA)      // __FALSE__
?gts(AAA, AAA)     // __TRUE__
?gts( AAA,AAA)     // __FALSE__

?gts("AAA", "AAA") // __FALSE__
?gts("AAA", " AAA")// __TRUE__
?gts(" AAA", "AAA")// __FALSE__
```

len**[Description Format]**

```
?len ([<Character String>])
```

[Function]

Return length of <Character String>.

[Explanation]

- In case of specifying particular character (exp parenthesis), use double quotation type <Character String>.

[Example]

```
?len(ABCDE)      // 5
?len( ABCDE )    // 7

?len("ABCDE")    // 5
?len(" ABCDE" )  // 5
```

les**[Description Format]**

```
?les ([<String 1>],[<String 2>])
```

[Function]

Compare <String 1> and <String 2>.

[Explanation]

- If <String 1> is less than or equal to <String 2>, return `__TRUE__`. Otherwise, return `__FALSE__`.
- In case of specifying particular character (exp parenthesis), use double quotation type <String 1> or <String 2>.

[Example]

```
?les(AAA,AAA)           // __TRUE__
?les(AAA, AAA)          // __FALSE__
?les( AAA,AAA)          // __TRUE__

?les("AAA", "AAA")     // __TRUE__
?les("AAA", " AAA")    // __FALSE__
?les(" AAA", "AAA")    // __TRUE__
```

lts**[Description Format]**

```
?lts ([<String 1>],[<String 2>])
```

[Function]

Compare <String 1> and <String 2>.

[Explanation]

- If <String 1> is less than <String 2>, return __TRUE__. Otherwise, return __FALSE__.
- In case of specifying particular character (exp parenthesis), use double quotation type <String 1> or <String 2>.

[Example]

```
?lts(AAA,AAA)           // __FALSE__
?lts(AAA, AAA)          // __FALSE__
?lts( AAA,AAA)          // __TRUE__

?lts("AAA", "AAA")     // __FALSE__
?lts("AAA", " AAA")    // __FALSE__
?lts(" AAA", "AAA")    // __TRUE__
```

nes**[Description Format]**

```
?nes ([<String 1>],[<String 2>])
```

[Function]

Compare <String 1> and <String 2>.

[Explanation]

- If <String 1> is not equal to <String 2>, return __TRUE__. Otherwise, return __FALSE__.
- In case of specifying particular character (exp parenthesis), use double quotation type in <String 1> or <String 2>.

[Example]

```
?nes(AAA,AAA)           // __FALSE__
?nes(AAA, AAA)          // __TRUE__
?nes( AAA,AAA)          // __TRUE__

?nes("AAA", "AAA")     // __FALSE__
?nes("AAA", " AAA")    // __TRUE__
?nes(" AAA", "AAA")    // __TRUE__
```


substr**[Description Format]**

```
?substr ([<Character String>], <Expression 1>, <Expression 2>)
```

[Function]

Extract substring from <Character String>.

[Explanation]

- Extract column from <Expression 1> to <Expression 2> of <Character String>.
- First character of <Character String> is assumed as 0 column.
- If the value of <Expression 1> exceeds length of <Character String>, empty string is extracted.
- If the value of <Expression 2> is 0, empty string is extracted.
- If the value of <Expression 2> exceeds length of <Character String>, it returns string from <Expression 1> to the end of <Character String>.
- If the value of <Expression 1> or <Expression 2> is negative value or exceeds maximum limits, error occurs.
- In case of specifying particular character (exp parenthesis), use double quotation type in <Character String>.

[Example]

```
?macro sample arg1
    ?arg1                                // "ABCDE"
    ?substr(?arg1, 0, ?len(?arg1))      // ABCDE

    ?substr(?arg1, 0, 1) // A
    ?substr(?arg1, 0, 2) // AB
?endm

sample "ABCDE"
```

15.6 Particular Macro

trigger**[Description Format]**

```
?trigger (<Trigger Character>)
```

[Function]

Change trigger character.

[Explanation]

- Question mark (?), period (.), dollar mark (\$) can be used as <Trigger Character>.

[Example]

```
?len("ABCDE")
?trigger(.)
.len("ABCDE")
```

Chapter 16 Macro Preprocessor Options

-D Define Macro

[Description Format]

`-D<Identifier>[=<Replacement Text>]`

[Function]

Defines <Replacement Text> to <Identifier>.

[Explanation]

- This is same as specifying #define directive in head of a source file.
- In case of specifying as "-D <identifier>", it means that <replacement text> is defined as 1.
- It is not possible to specify following cases.
 - When there is a blank character between <Identifier> and =
 - When there is a blank character between = and <Replacement Text>
- Macro Preprocessor processes -D, -U, -s in following order independently of specifying order.
 1. -D
 2. -U

-GN Specify File Name Used in Error Message

[Description Format]

`-GN <Filename>`

[Function]

Specify the filename which is used for error or warning message.

-I Specify Search Path for Include Files

[Description Format]

`-I <Path>`

[Function]

Add <Path> as the search path for include files.

[Explanation]

- This option adds the search path for include files specified in #include directive. <Path> cannot be omitted.
- This option can be specified multiple times. When multiple options specified, the paths are searched in the order in which the paths are specified.
- Searches are performed according to the path specified in this option when the filename is specified with a relative path. When a path is specified with absolute path, the specified file is read in without being searched.
- The file is searched in order of the following directories:

#include "file"

- (1) The directory of the source file.
- (2) The directory specified with -I option (When specified multiple times, the paths are searched in the order specified).
- (3) The include directory under the directory specified in the environment variable THOME900.

#include <file>

- (1) The directory specified with -I option (When specified multiple times, the paths are searched in the order specified).
- (2) The include directory under the directory specified in the environment variable THOME900.

-J Recognize the Kanji Code (Japanese version only)

[Description Format]

-J

[Function]

Recognizes the kanji code.

[Additional Note]

Do not use this option for English version.

-U Disable the Macro Definition

[Description Format]

-U<Identifier>

[Function]

Disables the macro definition specified with <Identifier>.

[Explanation]

- This is the same as specifying #undef directive in head of a source file.
- Macro Preprocessor processes -D, -U, -s in following order independently of specifying order.
 1. -D
 2. -U

-V Output Version Number

[Description Format]

-V

[Function]

Outputs the Macro Preprocessor version number to standard output.

[Explanation]

- This option cannot be included in files for which -f option is specified.

-e Create Error List File

[Description Format]

-e<Filename>

[Function]

Outputs all error messages to one file as a error list file.

[Explanation]

- This option outputs all errors and warnings that occur during Macro Preprocessor execution to the file specified in <Filename>.
- When a fatal error occurs, processing ends immediately and no error list file can be created.

-f Read Option List File

[Description Format]

-f<Filename>

[Function]

Reads the options from an option list file containing a startup option list.

[Explanation]

- Describes in advance in a text file the option to be specified and specified the file as <Filename>.
- It is possible to describe options on multiple lines within an option list file.
- This option can be specified multiple times to specify multiple option list files.

-g Create Debugger Information

[Description Format]

-g

[Function]

Output debug information to an output file.

-l Create a Macro Preprocessor List File

[Description Format]

-l

[Function]

Creates a macro preprocessor list file.

[Explanation]

- A macro preprocessor list file is generated by the file name which changed the suffix of the source file name to ".med".
- Macro Preprocessor outputs following information to a macro preprocessor list file.
 - Input source file and processed result.
 - Result of definition by replacement macro and numerical macro.
 - #include information.

-o Specify an Output Filename**[Description Format]**

<code>-o<Filename></code>

[Function]

Specifies the filename of the output file.

[Explanation]

- When this option is omitted, the output file is generated by the file name which changed the suffix of the source file name to ".asm".

-s Define Identifier of Variable Function**[Description Format]**

<code>-s <Identifier>[=<Value>]</code>
--

[Function]

Defines <Value> to <Identifier>.

[Explanation]

- This is same as specifying ?variable directive in head of a source file.
- If <Value> is omitted, it means that <Value> is defined as 1.
- It is not possible to specify following cases.
 - When the <Value> is not an integer
 - When there is a blank character between <Identifier> and =
 - When there is a blank character between = and <Value>

Chapter 17 Macro Preprocessor Limitation

Table below shows the Limitation of the Macro Preprocessor.

Table 17-1 Limitation of Macro Preprocessor

Item	Limitation
Number of lines of a source file	655360 lines
Number of characters of one line	65536 characters
Number of identifier characters	1024 characters
Number of characters of absolute path	1024 characters
Nest level of #if, #ifdef, #ifndef	63 level
Nest level of #include	255 level
Number of parameters of #define	127 parameters
Value that #line parameter can be specified	655360
Number of identifiers characters which is specified with option -s	1024 characters
Number of times that option -I can be specified	31 times
Number of times that option -D can be specified	65536 times
Nest level of ?macro, ?repeat, ?while, ?if	32 level
Repeat count of ?repeat	65535 times
Number of parameters of ?macro	127 parameters
Number of local symbols of ?macro	64 symbols
Value that ?substr parameter <Expression 1> and <Expression 2> can be specified	4294967295

Part 5 The Librarian

Chapter 18 The Librarian

18.1 Librarian Overview

The Librarian collects some relocatable object files created by the Compiler or Assembler to one file, and creates a library file.

When relocatable object files are registered in a library file and this library file is specified at linking, the linker can extract and link only required modules automatically.

Librarian Functions

Librarian has the following functions:

- Creation of library files
The librarian can create new library files.
- Registering modules in library files
Librarian can register the specified modules in an existing library file.
- Deletion of modules from library files
Librarian can delete specified modules from a library file.
- Updating modules in library files
Librarian replace the specified modules in a library file.
- Display identifier information from a modules in library files.
Librarian can output identifier information from the specified module in a library file to standard output.
- List names of modules in library files
Librarian can output the names and sizes of the modules in a library file to standard output.

18.2 Startup command

[Description Format]

```
tulib <Option> <Library_Filename> [<Filename>|<Module_name>]
```

[Explanation]

- Specify commands, <Option>, <Library_Filename>, and [<Filename>|<Module_name>] delimited with spaces.
- <Option> must be specify one of the following.
'-d' '-l' '-r' '-t'
- <Option> can specify multiple, however these option (-d, -l, -r, -t) are not specified in one time.
- <Library_Filename> is specified only one.
- Multiple file names (or module names) can be specified in [<Filename>|<Module_name>].

18.3 Input and Output Files

- Table below shows the suffixes used in the names of input and output files of Librarian.

Table 18-1 Input and Output files of Librarian

Suffix	File type	Classification
.lib	Library file	Input and output
.rel	Relocatable object file	Input

- **Library file**

This is a file that links multiple library files and relocatable object files. The librarian performs module registration, updates, and deletion, etc. on this library file. It is also possible to specify a library file as an input file.

This file is a binary format file that complies with the IEEE695 object module format.

- **Relocatable object file**

Relocatable object files output by the compiler or assembler are the input files for Librarian. The contents of such relocatable object files are registered in the library file as modules.

This file is a binary format file that complies with the IEEE695 object module format.

Chapter 19 Librarian Options

-V Output Version Number

[Description Format]

-V

[Function]

Outputs the Librarian version number to standard output.

[Explanation]

- When the Librarian is activated, startup messages such as the Librarian version number are output to standard output.
- This option cannot be included in files for which -f option is specified.

-d Delete a Module

[Description Format]

-d[<Suboption>] <Library_Filename> <Module_Name>

[Function]

Deletes modules from a library file.

[Explanation]

- Deletes modules specified <Module_Name> from a library file specified <Library_Filename>.
- Multiply <Module_Name> delimites with space.
- An error occurs if the modules specified <Module_Name> do not exist in the specified <Library_Filename>.
- When <Module_Name> is omitted, the librarian ends execution normally without processing a library file.
- This option cannot specify with -l or -r or -t option at a time.

Table 19-1 Suboptions of -d option

Suboptions	Function
v	Outputs the name of the deleted modules to standard output.

-e Create Error List File

[Description Format]

-e<Filename>

[Function]

Outputs all error messages to one file as a error list file.

[Explanation]

- This option outputs all errors and warnings that occur during librarian execution to the file specified in <Filename>. If <Filename> already exists, error messages will be added to the file.
- When a fatal error occurs, processing ends immediately and no error list file can be created.

-f Read Option List File**[Description Format]**

`-f<Filename>`

[Function]

Reads the options from an option list file containing a startup option list.

[Explanation]

- Describes in advance in a text file the option to be specified and specified the file as <Filename>.
- It is possible to describe options on multiple lines within an option list file.
- This option can be specified multiple times to specify multiple option list files.

-l Output Module Identifier Information**[Description Format]**

`-l <Library_Filename> [<module_List>]`

[Function]

Outputs identifier information for modules in a library file.

[Explanation]

- Specify module names delimited with space as <Module_List>.
- The following is output as identifier information in a module:
 - Module name
 - Size
 - External definition symbols
 - External reference symbols
- When <Module_List> is omitted, the information about all modules contained in a library file is output.
- An error occurs if the modules specified in <Module_List> do not exist in the specified library file.
- This option cannot specify with -d or -r or -t option at a time.

-r Create Library File and Register and Update Modules**[Description Format]**

`-r[<Suboption>] <Library_Filename> <Module_List>`

[Function]

Creates a library file and registers and updates modules.

[Explanation]

- The following files can specify as <Module_List>.
 - Relocatable object file
 - Library file
- This option registers and updates the modules specified in <Module_List> in the library file specified in <Library_Filename>.
- Specify module names delimited with space in <Module_List>.
- When the module which is included in file specified in <Module_List> exist in a library file specified in <Library_Name>, that module is updated.

- If the library file specified in <Library_Filename> does not exist, a message is output and a new file is created. The modules are then registered in the new library file.
- This option cannot specify with -d or -l or -t option at a time.

Table 19-2 Suboptions of -r option

Suboptions	Function
c	Message at creating a library file is not output.
u	Updates the modules, when the date and time of creation of file specified in <Module_List> is newer than the library file and the module is registered in the library file. When the file specified in <Module_List> does not exist in the library file, it can not add.
v	Outputs the name of the added or updated module to standard output.
w	Updates the modules, when the date and time of creation of file specified in <Module_List> is newer than the library file and the module is registered in the library file. When the file specified in <Module_List> exist in the library file, it is added to the library file.

-t Output Module Information

[Description Format]

```
-t[<Suboption>] <Library_Filename> [<Module_List>]
```

[Function]

Outputs a list of modules in a library file.

[Explanation]

- Specify module names delimited with space in <Module_List>.
- The module information contains the following information:
 - Module name
 - Module size in bytes (only when suboption v specified)
 - Module attribute (only when suboption v specified)
 - Module creation date and time (only when suboption v specified)
- An error occurs if the library file specified in <Library_Filename> do not exist.
- This option cannot specify with -d or -l or -r option at a time.

Table 19-3 Suboptions of -t option

Suboption	Function
v	Outputs size,attribute, and creation date and time in addition to the module name. When the suboption 'v' is omitted, only module names are output.

Part 6 The Object Converter

Chapter 20 The Object Converter

20.1 Object Converter Overview

The Object Converter is a utility tool for converting the absolute object files output by the Linker into an object format usable by an EPROM writer.

The user can select from five object formats: Intel HEX format (16-bit addressing), Intel extended HEX format (20-bit addressing), and Motorola S format (in 16-bit addressing, 24-bit addressing, and 32-bit addressing).

20.2 Startup command

[Description Format]

```
tuconv [<Option>] <Absolute_Object_Filename>
```

[Explanation]

- Specify command, <Option>, and <Absolute_Object_Filename> delimited with spaces.
- If the object format is omitted, Intel HEX format is selected.

20.3 Input and Output Files

- Table below shows the suffixes used in the names of input and output files of Object Converter.

Table 20-1 Input and Output files of Object Converter

Suffix	File type	Classification
.abs	Absolute object file	Input
.h16	Intel HEX	Output
.h20	Intel Extended HEX	Output
.s16	Motorola S Format (16-bit addressing)	Output
.s24	Motorola S Format (24-bit addressing)	Output
.s32	Motorola S Format (32-bit addressing)	Output
.o00 - .off	Overlay file	Output
User defined	Object converter list file	Output

- Absolute object file

This file is a object file of binary format that complies with the IEEE695 object module format, output by the linker. This file contain identifier and external symbols which are assigned absolute addresses.
- Object converter list file

Object converting information such as section allocations for the each output object file is output.

To output this file, the file name must be specified with option -lf. Using option -l outputs the information to standard output.

Chapter 21 Object Converter Options

-F Select Object Format

[Description Format]

`-F<Suboption>`

[Function]

Specifies the output object format.

[Explanation]

- The object formats are as follows:

Table 21-1 Object Formats

<Suboption>	Object format	Suffix
h16	Intel Hex format	.h16
h20	Intel extended Hex format	.h20
s16	Motorola S-16 format	.s16
s24	Motorola S-24 format	.s24
s32	Motorola S-32 format	.s32

- You cannot specify more than one object format at one time.
- If this option is omitted, it is recognized as option -Fh16 specified.
- If a numeric value of <Suboption> is omitted, the command is processed assuming a value 16 is specified. (-Fh = -Fh16, -Fs = -Fs16)

-P Specify Fill Value for Empty Area

[Description Format]

`-P[<Start_Address>],<Size>,<Value>,[<Output_Filename>]`

[Function]

Outputs <Size> area to the file specified with <Output_Filename> from <Start_Address>. At this time, the empty area is initialized using the specified <Value>.

[Explanation]

- When this option is used together with the option -ra, -rb, the empty area is filled after moving object by option -ra, -rb.
- Specify <Start Address> to define with 32-bit unsigned integer. When <Start Address> is omitted, the start address is addressed 0.
- <Size> specifies 32-bit unsigned integer in bytes. "k" following the numerical value represents kilo byte and "m" is mega byte.
- <Value> is 1 byte integer.
- When this option is used together with the option -ra, -rb, a different filename from the <Output Filename> which is specified by -ra, -rb option, cannot be specified.
- This option can be specified multiple times.

-V Output Version Number**[Description Format]**

```
-V
```

[Function]

Outputs the Object Converter version number to standard output.

[Explanation]

- When the Object Converter is activated, startup messages such as the Object Converter version number are output to standard output.
- This option cannot be included in files for which -f option is specified.

-c Specify a comment**[Description Format]**

```
-cc <Comment>
-cf <Filename>
```

[Function]

Specifies a comment to be inserted into an object file output by Object Converter.

[Explanation]

- <Comment> is a character string.
- <Comment> cannot include spaces.
- Multi-line comments can be written in the file which is specified in <Filename>.
- When converting a file to the Intel Hex format, the colon (:) cannot be used as the first character of a comment. If it specified, it is ignored and output a warning.

-e Create Error List File**[Description Format]**

```
-e <Filename>
```

[Function]

Outputs all error messages to one file as a error list file.

[Explanation]

- This option outputs all errors and warnings that occur during object converter execution to the file specified in <Filename>.
- When a fatal error occurs, processing ends immediately and no error list file can be created.

-f Read Option List File**[Description Format]**

```
-f <Filename>
```

[Function]

Reads the options from an option list file containing a startup option list.

[Explanation]

- Describes in advance in a text file the option to be specified and specified the file as <Filename>.
- It is possible to describe options on multiple lines within an option list file.
- This option can be specified multiple times to specify multiple option list files.

-l	Output an Object Converter List
----	---------------------------------

[Description Format]

```
-l[ <Suboption> ]
```

[Function]

Outputs an object converter list.

[Explanation]

- Output the following information as an object converter list.
 - CPU
 - Input file name
 - Output filename and object format
 - Section information (section name, allocation address, address offset, size)
 - Padding information (padding value, allocation address, size)
- The suboption types are as following.

Table 21-2 Suboptions of option -l

Suboptions	Function
No specification	Output an object converter list to standard output.
f <Filename>	Output a file specified in <Filename>. <Filename> cannot be omitted.

-o	Specify an Output Filename
-----------	-----------------------------------

[Description Format]

```
-o<Filename>
```

[Function]

Specifies the filename of the output file.

[Explanation]

- The suffix of the filename in <Filename> is not checked.
- When this option is omitted, the output file is generated by the file name which changed the suffix of the source file name according to the specified object format.
- When using this option together with the option `-ra` and `-rb`, this option is ignored.

-ra	Specify Object Output Range (address specification)
-----	---

[Description Format]

```
-ra [<Source Address>],[<Size>],[<Offset|Destination Address>],  
    [<Output Filename>]
```

[Function]

The object converter cut out the <Size> part object from the <Source Address> of the input file. The cut out part is moved to the address specified by <Offset|Destination Address>, and is output to the file specified by <Output Filename>.

[Explanation]

- Specify <Source Address> to define with 32-bit unsigned integer. When <Source Address> is omitted, this will be regarded as address 0.
- Specify <Size> to define with 32-bit unsigned integer. When <Size> is omitted, all objects after <Source Address> is output.
- Specify <Offset|Destination Address> as follows.
 - <Integer Value> Specifies the destination address.
 - +<Integer Value> Adds <Integer Value> as offset to <Source Address>.
 - <Integer Value> Subtracts <Integer Value> as offset to <Source Address>.
- <Output Filename> is omitted, the output file is generated by the file name which changed the suffix of the source file name according to the conversion format.
- When this option is specified multiple times and <Output Filename> is the same one, the respective options are collected into one output file.
- The object address is changed. However, the contents of the object is not changed.

-rb Specify Object Output Range (section specification)

[Description Format]

```
-rb <Section Name>,[<Size>],[<Offset|Destination Address>],
    [<Output Filename>]
```

[Function]

The object converter cut out the <Size> part object from the initial address of <Section Name> of the input file. The cut out part is moved to the address specified by <Offset|Destination Address>, and is output to the file specified by <Output Filename>.

[Explanation]

- Specify the section name which moves as <Section Name>.
- <Section Name> is the section name determined at linking. When the output section name is specified at linking, specify that output section name to <Section Name>.
- Specify <Size> to define with 32-bit unsigned integer. When <Size> is omitted or exceed the specified section size, only the specified section is output.
- Specify <Offset|Destination Address> as follows.
 - <Integer Value> Specifies the destination address.
 - +<Integer Value> Adds <Integer Value> as offset to the initial address of <Section Name>.
 - <Integer Value> Subtracts <Integer Value> as offset to the initial address of <Section Name>.
- <Output Filename> is omitted, the output file is generated by the file name which changed the suffix of the source file name according to the conversion format.
- When this option is specified multiple times and <Output Filename> is the same one, the respective options are collected into one output file.
- The object address is changed. However, the contents of the object is not changed.

Part 7 Error Message

Chapter 22 Error Message Format

22.1 Types of Error Message

There are the following three types of error message.

Warning

A warning is output when a compile result may become what a user does not mean. The compiler outputs a warning, but compiling work continues, and output file is generated.

Error

An error is output when syntax that violates the rules is detected.

Fatal Error

A fatal error is output when some kind of serious problem occurs under compiling and it is no longer possible to progress with source file compiling.

22.2 Error Message Format

Error messages take the following format:

<Filename> <Line Number> : <Tool>-<Type>-<Number> : <Message>

<Filename>	This is the name of the file in which the error occurred. <filename> is not output when the cause of an error is not related to a specific file.
<Line Number>	This is the number of the line in which the error occurred. Usually, it outputs to an error as which a filename is output.
<Tool>	This is the name of the tool (assembler etc.) in which the error is occurred.
<Type>	This is the error type. Fatal Error Warning
<Number>	This is the error number, described later. Fatal : 0 - 99 Error : 200 - 499 Warning : 500 - 999
<Message>	The message is a description of the error.

Chapter 23 Assembler Error Messages

23.1 Assembler Fatal Errors

<I/O Errors>

- 20: *Can't open "<filename>"***
The specified file cannot be opened.
- 21: *Can't close "<filename>"***
The specified file cannot be closed.
- 22: *Can't read "<filename>"***
The specified file cannot be read.
- 23: *Can't write "<filename>"***
The specified file cannot be written.
- 24: *Can't seek "<filename>"***
The specified file cannot be sought.

<Invocation Errors>

- 100: *No source file found in invocation***
No input source file is specified in the command.
- 101: *Illegal file specification***
The illegal filename is specified.
- 102: *File must be a disk***
You cannot specify a file other than a disk file.
- 103: *"<filename>" files are the same***
The same filename is specified more than once.
- 105: *Bad parameter syntax***
A parameter of an option is incorrect.
- 106: *Missing parameter "<option>"***
A parameter which should be specified for an option is missing.
- 107: *Illegal sub option in '-l'***
The suboption of '-l' is illegal.
- 109: *Unrecognized option "<option>"***
An invalid option is specified.
- 110: *Numeric constant out of range***
The numeric value is out of range.
- 111: *Can't nest a command file***
The option list file is nested.

<Execution errors>

- 152: *Illegal source file format***
The source file format is illegal.
- 154: *Internal object file error***
The object file is illegal.
- 155: *Too many expressions***
Too many expressions or expressions are too complex.

- 156: Optimization table overflow**
The optimization table overflowed.
- 158: String table overflow**
The string table overflowed.
- 160: Symbol table overflow**
The symbol table overflowed. Delete unused symbols, divide the assembly source file.
- 161: Out of memory**
The working memory area is insufficient.
- 162: Command line too long**
The command line or option list file exceeds the upper limit.
- 163: Too many "file" instructions**
The number of "file" directives of debugging information exceeds the upper limit.

23.2 Assembler Errors

- 200: Syntax error**
A syntax error occurred.
- 201: Attempt to divide by zero**
A divide by zero occurred.
- 202: Illegal numeric constant**
A numeric constant is illegal. Common causes are specifying characters which cannot be used in numeric constant.
- 203: Multi-defined symbol "<symbol>"**
The <symbol> is multi-defined.
- 204: Invalid relocatable expression**
There is an invalid relocatable expression. An absolute expression can be used for such as a shift amount of shift operation, using
- 205: Unbalanced parentheses**
Parentheses are unbalanced.
- 206: Invalid expression**
There is an invalid expression.
- 208: Illegal label or variable**
There is an illegal label or variable.
- 209: Illegal character string**
There is an illegal character string.
- 210: Not allowed public attribute**
A symbol for which public declaration is not allowed is used in public declaration.
- 212: Illegal SECTION directive**
There is an illegal SECTION directive.
- 215: No section definition**
Instructions are described without a section definition.
- 216: Invalid section attribute**
The section attributes or displacement of section directive is invalid.
- 217: Absolute section error**
Absolute section addresses overlap.
- 218: Illegal control**
There is an invalid control statement.

- 220: *Reference to multi-defined symbol***
A multiply defined symbol is referenced.
- 221: *Undefined symbol***
An undefined symbol is referenced.
- 222: *Absolute expression expected***
Except an absolute expression can not describe.
- 223: *Not allowed forward reference***
Forward references are not allowed.
- 225: *Not allowed section reference***
Section names can not reference.
- 226: *Illegal symbol reference***
There is an illegal symbol reference.
- 227: *Out of range for relative reference***
An offset of a relative branch instruction exceeds the effective range.
- 228: *Overflow in location counter***
A location counter overflows.
- 229: *Location counter can't point lower address***
The specified address is less than the current location counter.
- 230: *Operand type mismatch***
Types of operand do not match.
- 231: *Too few or many operands***
The number of operand is illegal.
- 232: *Section "<section_name>" does not exist***
The specified section does not exist.
- 234: *The nesting level is exceeded***
Include files are nested exceeding the maximum nesting level.
- 300: *Illegal operand value for CALLV***
The operand value of CALLV instruction is illegal.

23.3 Assembler Warning Errors

- 500: *Illegal string constant***
There is an illegal character constant description, or the number of character constant exceeds the upper limit. The exceeded part is truncated.
- 501: *Operand value is out of range***
The operand value is out of range. Unintended consequences may be obtained, because the only least significant bits are from the low order.
- 503: *Some optimizations lost***
Non-optimized labels remain.
- 504: *Invalid instruction in this section***
Machine instructions are described in the section which is not a code section.
- 505: *Invalid directive in this section***
The position described a directive is invalid. The directive is ignored.
- 506: *No END directive***
No END directive at the end of the source file. It is treated as if the directive exists.

507: *Text found after END statement*

There are source program after the END directive. The source programs after the END directive are ignored.

508: *Invalid value in ALIGN directive*

A value specified ALIGN directive is invalid.

509: *Duplicated MODULE directive*

MODULE directive has been redefined. The second and subsequent MODULE directives are ignored.

514: *Source file empty*

No source program is described in the specified source file.

515: *Illegal parameter*

The option is multiple specified. The option after second ignores.

516: *Illegal escape sequence*

There is an illegal escape sequence. The escape sequence is ignored.

517: *This section already has a different attribute*

A section of the same name has already been defined with a different type. The section definition is ignored and the preceding section is continued.

518: *The floating-point type is not correct.*

The floating type constant is illegal.

519: *Ignored option '<option>'*

-Nb option is multiple specified. The option after second ignores.

550: *Can't create a sort table, display symbols at random*

The symbol table cannot be sorted since the size of a source file is too large. Processing continues without sorting the symbol table.

Chapter 24 Linker Error Messages

24.1 Linker Fatal Errors

<I/O Errors>

- 20: *Can't open "<filename>"***
The specified file cannot be opened.
- 21: *Can't close "<filename>"***
The specified file cannot be closed.
- 22: *Can't read "<filename>"***
The specified file cannot be read.
- 23: *Can't write "<filename>"***
The specified file cannot be written.
- 24: *Can't seek "<filename>"***
The specified file cannot be sought.

<Invocation Errors>

- 100: *No source file found in invocation***
No input source file was specified in the command.
- 101: *Illegal file specification***
The illegal filename is specified.
- 103: *"<filename>" files are the same***
The same filename is specified more than once.
- 104: *Bad parameter syntax***
A parameter of an option is incorrect.
- 105: *Missing parameter "<option>"***
A parameter which should be specified for an option is missing.
- 106: *Illegal suboption in '-l'***
The suboption of '-l' is illegal.
- 108: *Illegal character "<character>"***
The illegal character is specified as an option.
- 110: *Unrecognized option "<option>"***
An invalid option is specified.
- 111: *Illegal numeric constant***
A numeric constant is illegal. Common causes are specifying characters which cannot be used in numeric constant.
- 112: *'-r' option requires '-o' option***
When '-r' option is specified, the output file must also be specified via '-o' option.
- 113: *Both '-r' and '-ng' are set***
When '-r' option is specified, '-ng' option cannot specify.

<Execution Errors>

- 120: *Bad object format in "<filename>" (<address>)***
The object file format in the input file is not correct.
- 121: *Illegal processor name in "<filename>"***
The processor name in the input file is illegal.

122: Illegal symbol class in "<section_name>" in "<filename>"

There is an illegal symbol class.

123: Illegal relocation type in "<section_name>" in "<filename>"

There is illegal information relating to section allocation in a file.

124: "<Symbol>" from "<filename>" already bound to an output section

The identifier is already bound to an output section.

130: Boundary "<constant>" not available in configured memory

There is incorrect information in a library file.

131: Fail to allocate "<count>" bytes for slotvec table

Sufficient memory could not be reserved in the work area due to insufficient memory.

133: Truncated "<section_name>" in "<filename>"

The file contains a truncated section.

134: Error(s). No output written to "<filename>"

Errors occurred so no output file was created.

139: Reloc entries out of order in "<section_name>" of "<filename>"

Relocation entries are abnormal.

141: Run is too large and complex

Memory allocation failed because the allocation specification was too complex. Use the incremental linking function to separate the levels and simplify linking.

<Link Command File Errors>**150: Syntax error**

A syntax error occurred.

151: Illegal section name or memory name

Memory name or section name is illegal.

152: Illegal address as origin or length

Memory start address or length specification is illegal.

153: Illegal memory specification

The specification in the memory definition part is illegal.

154: Multiple reference of a input section

A single section is referenced multiple times in a section definition part.

155: Illegal assignment

The assignment statement is illegal or too complex.

156: Semicolon required after expression

A semicolon is missing from the end of the assignment statement.

157: Bad fill value

Padding value is illegal in the '-F' option.

158: Multiple defined memory "<memory_name>"

The same user-defined memory is defined multiple times in the memory definition part.

160: Illegal output specification

Output specification is illegal in the section definition part. The main cause of this error is the multiple definition or incorrect sequence in the output specification 'org', 'align', 'len' and 'addr'.

161: OVERLAY section must BINDED

An address specification is required for an OVERLAY section.

163: Statement ignored

The statement is illegal.

166: Section not built "<section_name>"

The output section was not created. The main cause of this error is the inability to create the output section due to no memory area being allocated to the output section.

167: Missing Relocatable expression

There is a invalid expression. The main cause of this error is when undefined or unresolved output sections are specified by the operators 'org', 'addr' and 'sizeof'.

168: MEMORY segment overlap "<memory_name>" and "<memory_name>"

Memory areas specified in the memory definition part overlap.

169: Illegal operator in expression

The expression contains an invalid operator.

170: Can't set attributes "<attribute>"

The memory definition part contains memory attributes which cannot be specified. The cause of this error is characters other than 'RXWT' being specified, or one of these characters being specified more than once.

172: Can't nest a command file

The option list files are nested. This is because when a relocatable object file of an object format that is not IEEE695 is specified in a link command file, the Linker interprets that file as a link command file.

173: Illegal expression

The expression specification is illegal.

24.2 Linker Errors

201: "<section_name>" enters unconfigured memory at "<address>"

As a result of section allocation, section <section_name> was allocated in memory which was not defined in the memory definition part (at <address>).

The main causes are as follows:

- The area defined by the memory definition part is too small so it doesn't fit.
- Even though there is a memory definition part, the input section type memory is not defined.

202: Can't link "<section_name>" with different attribute

Sections of different types (code, data, etc.) cannot be combined.

203: Absolute sections "<section_name>" can't in SECTIONS

Absolute sections cannot be specified in the section definition part.

206: Section "<section_name>" overlap

The section overlaps another section.

207: Multiply defined "<symbol>" in "<filename>"

The symbol has been multiply defined.

209: Reference made to unresolved external symbol "<symbol>"

An unresolved symbol exists.

210: "<section_name>" at "<address>" won't fit into configured memory

<section_name> could not allocate to the specified area.

211: No space for "<section_name>" in "<memory_name>"

The memory allocated to the <section_name> is full.

214: "<section_name>" not yet allocated

The section has not been allocated.

- 217: Value of "<symbol>" in "<filename>" not fit in the object code**
The symbol value is not fit the object code.
- 218: DSECT "<section_name>" can't be given an owner**
Memory cannot be specified for a dummy section.
- 219: Multiply defined output section "<section_name>"**
The output section name has been multiply defined.
- 226: Can't allocate "<section_name>" to "<memory_name>"**
Cannot allocate memory in accordance with section attributes.
- 227: Attributes are mismatch between section and memory**
The section and its allocated memory have different attributes.
- 228: Illegal padding**
Padding was performed on an area which cannot be padded. Padding can only be performed on memory areas which have I attribute.
- 229: Making aux entry "<number>" for "<symbol>" out of sequence**
Auxiliary information was not created correctly.
- 231: Section "<section_name>" at "<address>" load value overflow. Truncated**
The output section relocation value does not fit in the object specified size. The higher order byte is discarded.
- 232: Symbol "<symbol>" size mismatch**
The extern declaration displacement and public declaration displacement are different.
- 233: Section "<section_name>" at "<address>" attempt to divide by zero**
A divide by zero occurred at location "<address>" in "<section>".

24.3 Linker Warning Errors

- 500: Absolute symbol "<symbol>" being redefined**
A symbol which allocated absolute address was redefined.
- 501: Symbol "<symbol>" from file "<filename>" being redefined**
A symbol was redefined.
- 504: Multiply defined symbol "<symbol>" from file "<filename>" has more than one size**
A symbol was multiply defined. The previous definition was for a different displacement.
- 505: "<number>" is not a power of 2**
The align operator parameter is not a power of two.
- 509: Useless MEMORY specification with '-r' option**
The memory definition part was ignored because '-r' option (perform incremental linking) was specified.
- 511: Unresolved external symbol "<symbol>"**
An unresolved external symbol exist.
- 513: Section "<section_name>" size lager than definition**
The output section size exceeds the size specified in the link command file.
- 514: All input files are LIBRARY files; no processor name exist**
As only library files were specified as input files the processor name could not be obtained.
- 515: Value is used what defined at "<filename>" as symbol "<symbol>"**
The symbol is multiply defined. The value of "<symbol>" specified in "<filename>" is used.

516: Value is used what lastly defined at LCF as symbol "<symbol>"

The symbol is multiply defined. The value which is specified as "<symbol>" at the last of the link command file is used.

517: Useless symbol definition with '-r' option

An assignment statement is defined in incremental linking.

518: Illegal parameter

A conflicting option is specified.

520: The predefined memory is overlapped "<memory_name>"

The predefined memory is multiply defined or overlapped.

523: Starting address of CODE or ROMDATA area is specified in 'addr'

The start address in the link command file is that of the CODE or ROMDATA area.

524: CODE or ROMDATA section is allocated in DATA area with 'org'

The input section of the CODE or ROMDATA type was allocated in the DATA area after specifying its address by 'org' of the link command file.

Chapter 25 Macro Preprocessor Error Messages

25.1 Macro Preprocessor Fatal Errors

<Command Line>

001: Invalid option '<option name>'

Invalid option is specified.

002: Unable to open option file <filename>

Option file cannot be opened.

003: Unable to open input file <filename>

The input file cannot be opened.

004: Unable to open output file <filename>

The output file cannot be opened.

005: Unable to open error output file <filename>

The error output file cannot be opened.

006: Unable to open list file <filename>

The list file cannot be opened.

007: Symbol not specified with '<option name>' option

The illegal symbol is specified with <option name> option.

010: Same filename <filename> specified

The same filename is specified in input files or output files.

011: Filename not specified with '<option name>' option

The filename has not been specified with <option name> option.

012: Source file not specified

Input filename has not been specified.

015: '<option name>' filename exceeds maximum limit

The filename whose number of character exceeds maximum limit is specified as <option name> parameter.

016: '<option name>' path name exceeds maximum limit

The path name whose number of character exceeds maximum limit is specified.

017: Source filename exceeds maximum limit

The input filename whose number of character exceeds maximum limit is specified.

018: Output filename exceeds maximum limit

The output filename whose number of character exceeds maximum limit is specified.

020: The parameter of option '<option name>' is not specified

No argument is specified for <option name>.

021: Extra source file specified

Two or more input files are specified.

<General>

040: Total number of characters in the line exceeds maximum limit

The number of characters of a line exceeds maximum limit. A line means the logical line after performed line splice.

041: Number of source lines exceeds limit

The number of lines which includes include file exceeds maximum limit.

042: *Out of memory*

Memory area cannot be allocated.

043: *Number of expansion exceeds maximum limit*

The number of expansions exceeds maximum limit.

<Preprocessor>

080: *'#endif' expected*

#endif directive in relation to #if, #ifdef or #ifndef is not specified.

081: *'#ifdef/#ifndef' expects an identifier*

An identifier is not specified with the #ifdef or #ifndef directive.

082: *Too many macro definitions*

The number of macro definitions exceeds maximum limit.

083: *Too many nested 'if'*

The number of nesting levels for #if directive exceeds maximum limit.

084: *Too many nested include files*

The number of nested #include files exceeds maximum limit.

085: *Unable to open include file <filename>*

The include file cannot be opened.

086: *Unexpected '<reserved word>'*

#else, #elseif or #endif directive is specified at illegal place.

087: *'#line' filename exceeds maximum limit*

The number of character of #line directive exceeds maximum limit.

088: *'#error' message text exceeds maximum limit*

The number of character of #error directive exceeds maximum limit.

<Macroprocessor>

None.

<Lexical>

160: *Unexpected EOF in block comment*

EOF occurs in block comment.

25.2 Macro Preprocessor Errors

<Command Line>

None.

<General>

240: *Syntax error*

A syntax error occurred.

241: *Number is invalid*

The number is not specified at the place where the number is specified.

242: *Right operand of shift operator has negative value*

Right operand of shift operator has negative value.

243: *Division/Remainder by zero*

Divisor or Remainder is 0, while evaluating constant expression.

244: *Too many characters in a character constant*

The number of character of a character constant exceeds maximum limit.

245: No character in a character constant

No character is specified as character constant.

246: Illegal expression

Illegal expression is specified.

<Preprocessor>**280: '#include' filename exceeds maximum limit**

The number of character of filename specified with #include exceeds maximum limit.

281: Syntax error in '#define'

The syntax of #define directive is not correct.

282: '##' cannot occur at the beginning of a macro definition

The place of ## operator is not correct. For example, the following description is not allowed.

```
#define AAA ## BBB    // Error
#define AAA CCC ## BBB // OK
```

283: '##' cannot occur at the end of a macro definition

The place of ## operator is not correct. For example, the following description is not allowed.

```
#define AAA BBB ##    // Error
#define AAA BBB ## CCC // OK
```

284: Formal parameter missing after '#'

It is specified except parameter after # operator.

285: #error : <string>

#error directive is specified.

286: Invalid line number for '#line'

A invalid line number is specified in the #line directive.

287: '#line' expects string as filename

A invalid character string is specified as filename in the #line directive.

288: '#undef' expect an identifier

A invalid character string is specified as parameter of the #undef directive.

289: Unexpected end of line

Corresponding parentheses are not specified in a same line.

290: '#include' expect a filename

A invalid character string is specified as filename in the #include directive.

291: Too many parameters for macro

The number of parameters exceeds maximum limit.

292: Too many or too few actual arguments in macro call <symbol>

The number of arguments different from definition is specified.

293: Redefinition of the reserved symbol '<reserved word>'

Reserved word cannot be redefined.

294: Redefinition of the predefined symbol '<reserved word>'

Defined macro cannot be redefined.

295: Redefinition of parameter name <parameter>

The parameter names is already used.

296: Unable to use '#undef' for the reserved symbol '<reserved word>'

Reserved word cannot be undefined.

297: Unable to use '#undef' for the predefined symbol '<reserved word>'

Defined macro cannot be undefined.

<Macroprocessor>

320: Too many local symbols or labels for macro

The number of local symbols or labels exceeds maximum limit.

321: Redefinition of symbol <symbol>

Symbol cannot be redefined. For example, a symbol used by ?macro cannot be redefined using ?variable.

322: '<reserved word>' is a reserved word

Reserved word cannot be redefined.

323: Redefinition of parameter name <parameter>

The parameter name is already used.

324: Redefinition of local symbol name or label name <symbol>

<symbol> cannot be redefined.

325: Macro function nest overflow

The number of nesting levels exceeds maximum limit.

326: Too many parameters for macro

The number of parameters exceeds maximum limit.

327: Too many or too few actual arguments in macro call <symbol>

The number of arguments different from definition is specified.

328: Reserved word '<reserved word>' is not made a parameter

Reserved word cannot be specified as a parameter.

329: Reserved word '<reserved word>' is not made a local symbol or a label

Reserved word cannot be specified as a local symbol or a label.

330: Macro call <symbol> should be in a new line

The character except space is specified before macro call.

331: Repeat value <expression> out of range

The number of arguments of ?repeat statement exceeds maximum limit.

332: Macro function <symbol> should be in a new line

Macro function cannot be specified in multi line.

333: Valid trigger character expected

The illegal trigger character is specified.

334: '?elseif' after '?else'

?elseif statement used after ?else statement.

335: '?else' after '?else'

?else statement used after ?else statement.

336: '?elseif/?else' without '?if'

?elseif or ?else statement used without previously using ?if statement.

337: '?endif' without '?if'

?endif statement used without previously using ?if statement.

338: '?restrict_macro' has already been specified

?restrict_macro statement cannot be nested.

339: '?endres' has been specified without '?restrict_macro'

?endres statement used without ?restrict_macro statement.

340: '?endres' has not been specified

?endres statement used without ?restrict_macro statement.

341: Unable to use architecture specific function '<macro function>'

Invalid function is used.

343: Number of macro expansion exceeds maximum limit

The number of macro expansion exceeds maximum limit.

344: No identifier after the trigger character

Trigger character needs identifier after it.

345: Unexpected identifier <symbol> is specified after the trigger character

Invalid identifier is specified after trigger character.

346: Function macro definition cannot be nested

Function macro cannot be nested.

<Lexical>

None.

25.3 Macro Preprocessor Warning Errors

<Command Line>

500: Extra source file ignored

Two or more input files are specified. The first input file specified is valid.

501: Duplicate options '<option name>' have been specified, only the first option is valid

The options are specified more than once. The first option specified is valid.

503: '<option name>' is ignored because list option is not given

<option name> is ignored because '-l' option is not specified.

<General>

540: Expression greater than 32bit length, excess ignored

The result of expression exceeds 32bit length. The exceeded part is truncated.

541: Decimal number has octal prefix

Octal number specification is illegal.

<Preprocessor>

580: Unexpected characters following directive '<directive>'

Extra characters are specified after <directive>. That characters is ignored.

581: Macro <symbol> redefined

The macro is redefined.

582: Length of the replacement text exceeds maximum limit, excess truncated

The number of characters in the replacement string exceeds maximum limit.

<Macroprocessor>

None.

<Lexical>

660: Identifier too long, excess truncated

The identifier whose number of character exceeds maximum limit is specified. The excess part is ignored.

Chapter 26 Librarian Error Messages

26.1 Librarian Fatal Errors

<I/O Errors>

- 20: *Can't open "<filename>"***
The specified file cannot be opened.
- 22: *Can't read "<filename>"***
The specified file cannot be read.
- 23: *Can't write "<filename>"***
The specified file cannot be written.
- 24: *Can't seek "<filename>"***
The specified file cannot be sought.
- 25: *Can't creat "<filename>"***
The specified file cannot be created.
- 26: *Can't creat temp file***
A work file cannot be created.
- 27: *Can't open command file "<filename>"***
The command file cannot be opened. The main cause of this error is that the file does not exist.
- 28: *"<filename>" is not reading permitted***
The specified file does not have read permission.
- 29: *"<filename>" is not writing permitted***
The specified file does not have write permission.

<Invocation Errors>

- 101: *Illegal file specification***
The illegal filename is specified.
- 110: *Unrecognized option "<option>"***
An invalid option is specified.
- 115: *usage: tulib -[drtl][vuc] files***
Required command parameters were not specified.
- 116: *One of [drtl] must be specified***
A required option was not specified. One of the options d, t, r or l must be specified.
- 117: *Only one of [drtl] allowed***
Incompatible options were specified. The options d, t, r and l can be specified at one time.
- 118: *Cannot use option in command file***
An option which cannot be used in a option list file is specified.
- 119: *Target processor is different***
The target MCU does not match in the object files or library files.
- 125: *Option '-r' is libraryfile or objectfile***
It is specified except a library file or object file with '-r' option.
- 126: *"<filename>" not in library format***
The specified file is not in library format. The main cause of this error is specifying except library file as a library file.

127: "<filename>" internal header generation error

An internal header error occurred in the specified file.

135: Dynamic storage allocation failure

Working memory cannot be reserved. Divide the library files into smaller files, and then retry.

136: Time of "<filename>" is broken

The time read from the specified file is incorrect. The file contents are invalid, so re-create is required.

26.2 Librarian Errors

240: Cannot open "<filename>"

The specified file cannot be opened.

241: "<filename>" not found

The specified file cannot be founded.

243: Status of objectfile is *ERROR*

The execution status of the specified object file is ERROR.

244: Status of objectfile is *WARNING*

The execution status of the specified object file is WARNING.

245: "<module>" does not exist in library file

The specified module does not exist in the library file.

247: "<filename>" is not writing permitted

The specified file does not have write permission.

248: Cannot make library file without module

A module was not specified. A library file cannot be created without a module.

250: "<filename>" is not an object file

The specified file is not an object file. The main cause of this error is specifying except object file as an object file.

26.3 Librarian Warning Errors

None.

Chapter 27 Object Converter Error Messages

27.1 Object Converter Fatal Errors

<I/O Errors>

- 20: *Can't open "<filename>"***
The specified file cannot be opened.
- 21: *Can't close "<filename>"***
The specified file cannot be closed.
- 23: *Can't write "<filename>"***
The specified file cannot be written.
- 24: *Can't seek "<filename>"***
The specified file cannot be sought.

<Invocation Errors>

- 100: *No source file found in invocation***
No input source file is specified in the command.
- 101: *Illegal file specification***
The illegal filename is specified.
- 102: *File must be a disk***
You cannot specify a file other than a disk file.
- 103: *"<filename>" files are the same***
The same filename is specified more than once.
- 105: *Bad parameter syntax***
A parameter of an option is incorrect.
- 106: *Missing parameter "<option>"***
A parameter which should be specified for an option is missing.
- 107: *Illegal sub option***
The suboption is illegal.
- 109: *Unrecognized option "<option>"***
An invalid option is specified.
- 110: *Can't nest a command file***
The option list file is nested.
- 111: *Not supported option "<option>"***
Not supported options are specified.
- 112: *Illegal output file "<filename>"***
The output file specified is illegal. When '-P' option is used with '-ra' or '-rb' option, the output file name must conform to any file name specified with '-ra' or '-rb' option.
- 113: *Illegal numeric constant***
Illegal numeric value is specified.
- 114: *Ambiguous point in block definition***
The address specified with '-ra' option has ambiguous points. The source address overlap the other section like have OVERLAY attribute, the specified with '-rb' option is valid.
- 115: *Can't find section "<section_name>"***
No section specified with '-rb' option.

116: *Can't find object in the area*

No object within the range specified by '-ra' option.

150: *Not an absolute object format*

The input file is not an absolute object format file.

151: *Bad object format*

The object format of the input file is invalid.

152: *Too large address*

An address in the input file exceeds maximum limit for the specified format. Otherwise, the address specified with '-ra' or '-rb' option has negative value.

153: *Load address overflow*

An address exceeded maximum limit for the specified format.

154: *Out of memory*

The working memory area is insufficient.

155: *Address overlap in "<filename>"*

Address specified in the output range overlaps.

27.2 Object Converter Errors

None.

27.3 Object Converter Warning Errors

500: *Illegal character in Comment*

A comment contains an invalid character.

501: *Comment too long*

A comment is too long.

515: *Illegal parameter*

An invalid parameter is specified.

516: *Ignored option "<option>"*

The option specification is ignored.

History

Issue	Date	Update
1st Edition	7 Jan, 2009	1st Edition

TLCS-900 Assembler Reference [1st Edition]

The Date of Issue: 7 Jan, 2009

TDE122-01