TOSHIBA

TLCS-870 Family Language Tools Operation Guide

4th Edition

Toshiba Corporation Semiconductor Company

- 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 patent or patent rights of TOSHIBA or others. (W06AE-01)

- The names of the companies, the systems and the products described in this document may be the trademarks of each company. (W07AE-01)

- The information contained herein is subject to change without notice. (W11AE-01)

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.

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.

Internet Information Service

The latest information on Toshiba microcomputer development system products is available on "Toshiba Microcomputer Development System Website" at:

http://www.semicon.toshiba.co.jp/mctool/index_e.htm

You can find the following information and more on this Website.

- Topics
- Product Release Schedule
- Product Introduction
- Latest Versions
- FAQ for Prospective Customers
- Product List for Each MCU

Contents

Part 1 Getting Started

Chapter 1 Setting Up Execution Environment

When using command line by the Command-Prompt of MS-Windows, it is necessary to perform the following environmental setup.

[THOMExxx] ❑ Set the home directory (the directory in which the software programs have been installed). The environment variables THOME and THOMExxx are used to set the home directory.

(*) THOMExxx is as follows:

THOME870C	TLCS-870/C,C1 Series
THOME870X	TLCS-870/X Series
THOME870	TLCS-870 Series

These environment variables are used for

The driver starts up each program

Searches for standard include files

[PATH] ❑ Set the bin directory in the directory where the software programs have been installed.

[TMP] ❑ Set the work directory for the programs. (Programmer can specify arbitrary directory to environment variables TMP.)

[NOTE] ❑ If the specified work directory is nonexistent or does not have sufficient space, an error may occur when executing the program.

[Example] ❑ When software installs in C:\TOSHIBA directory, the method of setting up an environment variables with a command line is as follows.

```
C>set THOME870C=c:\toshiba
C>set TMP=c:\work
C>path=c:\toshiba\bin
```

[NOTE] Do not insert spaces before and after the equal sign (=).

Select [My Computer]-[Control Panel]-[System]-[Advanced]-[Environment Variables] when using Windows; then set the following environment variables as either system environment variables or user environment variables according to the usage.

path “C:\Program Files\TOSHIBA\T870C\bin”
THOME870C C:\Program Files\TOSHIBA\T870C

Part 2 Tools

Chapter 1 CC Driver

1.1 Introduction

CC Driver controls the whole C Compiler process.

- Create an absolute object file from the specified source files in C and in assembly language.
- Link one or more relocatable files with standard C library files and other routines required for the C program to run to create an executable absolute object file for the target system.
- When compiling the C program, CC Driver activates each of the following programs in sequence:
  - Parser
  - Code generator

  The above sequence creates an assembler source file.
  - Assembler

  The above creates a relocatable object file.
  - Linker

  The Linker links the relocatable object files with a library to create an absolute object file.

Note
- The link command file and the start up routine must be specified by user.
- The parser and code generator cannot be run independently of CC Driver.

1.2 Startup command

Command

```
<cc_driver_name> [<option_list>] <source_filename_list>
```

Description
- Specify [<option_list>] delimited with spaces.
- Specify filenames delimited with spaces in the source filename list.
- See the Part3, Chapter3 for details of options.
- CC Driver are provided for the each MCU. The names of the drivers are as follows:

MCU	CC Driver Names
TLCS-870/X Series	cc870x
TLCS-870 Series	cc870
TLCS-870/C,C1 Series	cc870c

1.3 Input Files

CC Driver determines the type of file and the appropriate process to perform from the filename extension. Following table shows the types of input files processed by CC Driver and the filename extensions used by the driver to determine which process is applicable.

Extension	Type of input file
.c	C language source file
.i	C language source file (output from Macro Preprocessor)
.mac	Macro Preprocessor source file
.asm	Assembler source file
.rel	Relocatable object file
.lib	Library file
.lcf	Link command file

The following describes how CC Driver processes the respective input files.

- C language source files

  Compiles C language source files to create a relocatable object file, which is linked with other files.
- Macro Preprocessor source files

  Activates Macro Preprocessor and Assembler to create a relocatable object file, which is linked with other files.
- Assembler source files

  Activates Assembler to create a relocatable object file, which is linked with other files.
- Relocatable object files

  These files are passed to the Linker during linking.
- Library files

  These files are also passed to the Linker during linking
- Link command files

  These files are passed as the command files to the Linker.
- Unidentifiable files

  Files with extensions not listed above, or files without extensions, are all passed to the Linker.

1.4 Output files

The files output by each of the tools activated by CC Driver take names with the extensions shown as follows: If a filename is specified using the option for specifying the name of an output file, the output file is created with the specified filename.

Extension	Type of output file
.i	Macro Preprocessor output file (-P option specified)
.asm	Assembler source file (-S option specified)
.lst	Assembler list file (-l option specified)
.rel	Relocatable object file
.abs	Absolute object file
.map	Link map file (-l option specified)

1.5 List of Options

Option	Function
-#	Display Path and Options
-A	Compilers in accordance with ANSI specification
-D	Define Macro
-E	Preprocessor Output
-F	Fill Value
-I	Directories for Include Files
-J	Select Kanji Mode (Japanese version only)
-L	Directories for Input Files
-Mi	No Macro Processing
-Nc	Select Memory Style(TLCS-870/C,C1 only)
-O	Select Optimization Level
-P	Run Preprocessor Only
-S	Create Assembler Source
-T	Set Environment Variable
-U	Undefine Macro
-V	Output Version Number
-W	Pass Options to Tool
-XE	Ignore Escape Sequence
-XF	No Deletion of Assembler Source
-XS	Code Size Optimization
-Xec	Specify Default Function Qualifier
-Xi	Run Macro Processor
-Xns	Suppress Optimization (Integration of Stack Freeing)
-Xr	Set Default Function Modifier (_adecl)
-Xub	Select Unsigned Bit FIeld Members
-Xuc	Select Unsigned Characters
-Xw	Change bit field assignment order
-ZA	Set Default Section Size (area)

-ZC	Set Default Section Size (const)
-ZD	Set Default Section Size (data)
-ZT	Set Default Section Size (code)
-Za	Set Default Section Name (area)
-Zc	Set Default Section Name (const)
-Zd	Set Default Section Name (data)
-Zi	Set Default Section Name (io)
-Zt	Set Default Section Name (code)
-c	Create Object but do not Link
-e	Create Error List File
-f	Read Options from a File
-g	Create Debugger Information at assemble phase
-g	Create Debugger Information at linking phase
-l	Create List File
-m	Select Memory Model (TLCS-870/X only)
-o	Set Output Filename
-s	Define SET Symbol
-u	Delete All Predefined Macros
-u	Record Undefined Symbol
-w	Select Warning Level

1.6 Example Commands

Example Compiling a single source file

```
cc870c -Nc1 s1.c startup.rel sample.lcf
```

The C Compiler compiles the C language source file 's1.c', creates an assembler source file, assembles it, and creates a relocatable object file. Subsequently, CC driver links with the start up program(startup.rel) and standard library according to the link command file(sample.lcf). Thus the absolute object file is created. The absolute object file takes the name 's1.abs' comprised of the name of the original C language source file (s1.c) with extension '.abs'. The relocatable object file is retained even after final compiling. The assembler source file created during the compiling process is deleted unless the '-S' option is specified.

Example Compiling multiple source files

```
cc870c -Nc1 s1.c s2.c s3.c startup.rel sample.lcf
```

The C Compiler compiles and assembles the three C language source files 's1.c', 's2.c' and 's3.c'. Subsequently, CC driver links with the start up program and standard library according to the link command file. Thus the absolute object file ‘sl.abs’ is created. The absolute object file takes the name 's1.abs', which is comprised of the name of the first specified file on the command line as input files with extension '.abs'.

Example Compiling files with different extensions

```
cc870c -Nc1 -os8.abs s1.c s2.c s3.c s4.asm s5.rel s6.lib
s7.lcf
```

The C Compiler compiles and assembles 's1.c', 's2.c', and 's3.c', and assembles 's4.asm'. Subsequently, CC driver links with five relocatable object files of ‘sl.rel’-’s5.rel’, the library file ‘s6.lib’ and standard library according to the link command file ‘s7.lcf’. The name of the created absolute object file is 's8.abs' (- os8.abs).

Example Compile and assemble only. Create a file of error messages.

```
cc870c -Nc1 -c -e errorlst.err s1.c s2.c s3.c
```

This command line compiles and assembles the source files (-c) and creates a file of only error messages (errorlst.err) (-e errorlst.err).

Example Creating a file with information for source level debugging.

```
cc870c -Nc1 -g s1.c s2.c s3.c
```

This command line compiles the 3 source files, assembles, and links the appropriate modules and creates the absolute object file 's1.bas' with information for source level debugging (-g).

Example Setting preprocessor macros and an include file search path.

```
cc870c -Nc1 -D ROOT -D CASE=2 -I /usr/inc s1.c s2.c s3.c
```

This command line compiles, assembles, and links the source files. The files are compiled after setting the preprocessor macros (-D ROOT -D CASE=2) and the include file search path (-I/usr/inc).

Chapter 2 Assembler

2.1 Introduction

- Assembler converts assembly language programs into relocatable object programs.

2.2 Startup command

Command

```
<assember_name> [<option_list>] <source_filename>
```

Description

- Specify commands, options, and the name of a source file delimited with spaces.
- See the Part3, Chapter3 for details of options.
- Specify only one source program.
- Assembler is provided for each MCU. The names of Assemblers are follows:

MCU	Assembler
TLCS-870/X Series	asm870x
TLCS-870 Series	asm870
TLCS-870/C Series	asm870c

2.3 Input Files

Extension	File type
.asm	Assembly language file

- Assembler processes files containing source statements written in assembly language.

2.4 Output Files

Extension	File type
.rel	Relocatable object file
.lst	Assemble list file

- Relocatable object files are the result of assembling. These files are in binary format conforming to IEEE695.
- Assemble list files include the assemble list, symbol list, cross-reference list, etc. The -l option must be specified for these files to be output.

2.5 List of Options

Option	Function
-I	Directories for Include Files
-J	Select Kanji Mode (Japanese version only)
-O	Select Optimization Level
-V	Output Version Number
-XE	Ignore Escape Sequence
-Xt	Syntax Check Only
-e	Create Error List File
-f	Read Options from a File
-g	Create Debugger Information at assemble phase
-l	Create List File
-o	Set Output Filename
-w	Select Warning Level

2.6 Example Commands

Example Assemble a source file

```
asm870C sample.asm
```

Assembler processes the assembler source file 'sample.asm' and outputs an object file 'sample.rel'.

Example Assemble a source file and output an assemble list

```
asm870C -lx sample.asm
```

This command outputs the object file 'sample.rel' and the assemble list file 'sample.lst'. The list includes a cross-reference (-lx).

Example Assemble, but create only an error message file

```
asm870C -e errorlst.err -Xt -I/usr/inc sample.asm
```

This command creates the error message file 'errorlst.err' (-eerrorlst.err) without creating the object file or assemble list file. The assembling is executed after setting the search path for include files (-I/usr/inc).

Chapter 3 Linker

3.1 Introduction

The linker links the multiple relocatable object files created by compiling and assembling separate modules of a program. The linker also allocates the modules to memory and creates an absolute object file.

- All sequences performed in linking are specified using a Link Command file. The Link Command file allows details such as the following to be specified.
  - Specify the target memory configuration
  - Specify the order in which sections are linked
  - Specify the address sections are to be allocated to and the address range
  - Define and redefine public symbols.
- The required library modules are extracted from library files during linking.
- Special attributes can be applied to sections, such as "no substance" or "overlay".

3.2 Startup command

Command	tulink [<option_list>] <file_list>

Description
- Specify commands, option list, and the file list delimited with spaces.
- See the Part3, Chapter3 for details of options.

3.3 Input Files

Extension	File type
.rel	Relocatable object files
.lib	Library files
.lcf	Link Command files

- Relocatable object files

  These object files, which are relocatable object modules, are input and output by the linker. The symbols in these modules are assigned relative addresses from the beginning of each section. Some external symbols, however, remain without addresses. The relocatable object files are binary data files conforming to the IEEE695 format.

- Library files

  The library files consist of multiple relocatable modules collected into one by the Librarian (TULIB). The required modules are extracted from these files during linking. They are binary data files conforming to the IEEE695 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.

3.4 Output Files

Extension	File type
.abs	Absolute object files
.map	Link MAP files

- Absolute object files

  The absolute object files are output by the linker. The internal and external symbols in these modules have absolute addresses assigned to them and the modules can therefore be written to ROM. These files are binary data files conforming to the IEEE695 format.

- Link MAP files

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

3.5 List of Options

Option	Function
-F	Fill Value
-L	Directories for Input Files
-T	Set Environment Variable
-V	Output Version Number
-e	Create error list file
-g	Create Debugger Information at linking phase
-l	Create List File
-o	Set Output Filename
-r	Select Incremental Linking
-u	Record Undefined Symbol
-w	Select Warning Level

3.6 Example Commands

Example Link 'sample1.rel' and 'sample2.rel' according to memory mapping, etc., in the link command file 'sample.lcf'.

```
tulink sample.lcf sample1.rel sample2.rel
```

If no link command file stipulating memory allocation is specified, the linker starts allocating memory from address 0.

Example As above, but the names of input files are specified in the link command file 'sample.lcf'.

```
tulink sample.lcf
```

Example As above, but null output sections, if they exist, are filled with zeros.

```
tulink -F0x0000 -lg -lm -ll sample.lcf
```

A link list is also output (-l). The link list contains public symbols (-lg), local symbols (-ll), and a link map (-lm).

Chapter 4 Macro Preprocessor

4.1 Introduction

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.

4.2 Startup command

Command	tumpp [<option_list>] <source_filename>

Description

- Specify commands, options, and the name of a source program delimited with spaces.
- See the Part3, Chapter3 details of options.
- Specify only one source program.

4.3 Input Files

Following table shows the extensions used in the names of input files of Macro Preprocessor.

Extension	File type
.mac	Macro Preprocessor source file

- Macro Preprocessor source file is source file which is described using Macro Preprocessor language.

4.4 Output Files

Following table shows the extensions used in the names of output files of Macro Preprocessor.

Extension	File type
.asm	Assembler source file
.med	Macro preprocessor list file

- Assembler source file is source file which processing result of Macro Preprocessor.
- Macro Preprocessor list file is list file which includes input source file, processed result, symbol list, cross reference.

4.5 List of Options

Option	Function
-D	Predefine name as a macro
-GN	Set the file name which used for error message
-I	Specify the search path of include file
-J	Enables SJIS character
-U	Cancel any previous definition of name
-V	Show version number
-e	Create error message file
-f	Read specified option file
-g	Process debugging information
-l	Output Macro preprocessor list file
-mf	Specify the list file name
-ms	Specify the expanding history to simple
-o	Specify the output file name
-s	Specify the variable name and the value

4.6 Example Commands

Example Preprocessing and macroprocessing to a source file.

```
tumpp sample.mac
```

The Macro preprocessor source file ‘sample.mac’ is processed with Macro Preprocessor, and Assembler source file ‘sample.asm’ is outputted.

Example Outputting a macro preprocessor list file.

```
tumpp -l sample.mac
```

This command outputs Assembler source file ‘sample.asm’ and macro preprocessor list file ‘sample.med’.

Example Performs macro preprocessing and outputs an error message file

```
tumpp -e errorlst.err sample.mac
```

The error messages are output to the file 'errorlst.err' (-e errorlst.err).

Example Setting macro perprocessor macros and search path for include files and macro library

```
tumpp -s MACROID=2 -I/usr/inc sample.mac
```

This command sets Macro preprocessor macros (-s MACROID=2), the search path for the include files, and Macro library (-I/usr/inc) and executes macro processing.

Chapter 5 Librarian

5.1 Introduction

The librarian is a utility that collects the object files created for individual functions by the compiler and assembler into one file for easier management.

- Those object files that make up a program and which have been completed and will undergo no further change can be cataloged in a library file. This library file can then be specified at linking for the linker to automatically extract and link the required modules.
- Collecting general-purpose modules in a library file can facilitate the development of other programs.
- The librarian has the following functions:
  - Creation of library files
  - Cataloging modules in library files
  - Deletion of modules from library files
  - Updating modules in library files
  - Display symbol information from a modules in library files.
  - List names of modules in library files

5.2 Startup command

Command	tulib [<option_list>] <library_filename>[<module_list>]

Description

- Specify commands, options, library file names and module lists delimited with spaces.
- Always specify one and only one of the following options.
  '-d' '-l' '-r' '- t'.
- See the Part3, Chapter3 for details of options.
- Specify only one library file.
- Specify the modules to be processed by options in the module list.

5.3 Input Files

Extension	File type
.lib	Library files
.rel	Relocatable object files

- Library files

  The whole library or modules in the library can be extracted and recorded and updated in the output library file.

- Relocatable object files

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

5.4 Output Files

Extension	File type
.lib	Library files

- Library files

  Library files are created by linking multiple object modules in a special format. The librarian catalogs, updates, and deletes modules in a library file.

5.5 List of Options

Option	Function
-T	Set Environment Variable
-V	Output Version Number
-d	Delete a Module
-f	Read Options from a File
-l	Output Module Symbol Information
-r	Replace Modules
-t	Output Module List

Some of the option parameters specifying processing of the library file can take the following sub options.

Sub	Target	Function
c	-r	Suppresses message output
u	-r	Updates only recent modules
v	-d -r -t	Outputs details of librarian processing
w	-r	Updates only recent modules (newer than modules in LIB file)

5.6 Example Commands

Example Collect 'sample1.rel', 'sample2.rel', and 'sample3.rel' in new library file 'sample.lib'.

```
tulib -r sample.lib sample1.rel sample2.rel sample3.rel
```

This command creates a new library file, 'sample.lib'.

Example Add 'sample4.rel' to library file 'sample.lib' and replace 'sample3.rel'.

```
tulib -rv sample.lib sample4.rel sample3.rel
```

Option '-rv' sends the newly recorded and updated module names to standard output.

Example Replace or add 'sample1.rel' in library file 'sample.lib'.

```
tulib -rvu sample.lib sample1.rel
```

Option '-ru' updates 'sample1.rel' only when it is later than the one in the library file.

Example Outputs a list of modules in an updated library file.

```
tulib -tv sample.lib
```

The module list is output as follows:

```
module1      279 relocatable Mar 07 22:05 1992
module2      426 relocatable Mar 07 22:02 1992
module3      158 relocatable Mar 07 22:03 1992
module4      353 relocatable Mar 07 22:04 1992
```

Option '-tv' displays the size and creation date in addition to the name of the module.

Example Delete 'module2' (sample2.rel) and 'module3' (sample3.rel) from library file 'sample.lib'.

```
tulib -d sample.lib module2 module3
```

After deletion, modules 'module1' (sample1.rel) and 'module4' (sample4.rel) remain in the library file.

Example Output information on the symbols in the remaining modules, 'module1' (sample1.rel) and 'module4' (sample4.rel).

```
tulib -l sample.lib module1 module4
```

The output symbol list is as follows:

```
MODULE       INFORMATION
Name  Size   Type
module1      279
module4      353
PUBLIC SYMBOL(S) :
module1      init
module4      table
EXTERN SYMBOL(S)
module1      No Symbol
module4      calc
             init
```

Here, the information is the same for all modules in the library file. The same result would therefore be gained from the following command.

```
tulib -l sample.lib
```

Chapter 6 Object Converter

6.1 Introduction

The object converter is a utility 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, Intel extended HEX format, and Motorola S format (in 16-bit addressing, 24-bit addressing, and 32-bit addressing).
- Intel and Motorola format comments can be embedded.

6.2 Startup command

Command

```
tuconv [<option_list>] <abs_object_filename>
```

Description

- Specify commands, options, and object file names delimited with spaces.
- See the Part3, Chapter3 for details of options.
- Intel HEX format is selected if no output object format is specified.
- The output object file can be divided into two or more files.
- Default format is the Intel HEX format.
- Specify only one absolute object file.

6.3 Input Files

Extension	File type
.abs	Absolute object file

- Absolute Object Files (Input)

  Absolute object files are output by the linker. These modules contain local and public symbols with absolute addresses and can be written to ROM. These files are binary data files in the IEEE695 format.

6.4 Output Files

Extension	File type	Classification
.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
Use defined	Object converter list file	Output

- Object converter list file (output)

  Object converting information such as section allocations for the respective output object files is output. To output this file, the file name must be specified with option '-If'. Using option '-I' outputs the information to the console.

6.5 List of Options

Option	Function
-F	Select Object Format
-P	Fill Value
-T	Set Environment Variable
-V	Output Version Number
-c	Specify a comment
-e	Create error list file
-f	Read Options from a File
-l	Create List File
-o	Set Output Filename
-ra	Set Object Output Range (address specification)
-rb	Set Object Output Range (section specification)

6.6 Example Commands

Example

```
tuconv sample.abs
```

This command processes the linked absolute object file 'sample.abs' to output an object file 'sample.h16' in Intel HEX format.

Example

```
tuconv -Fh20 -cc This_is_sample sample.abs
```

This command outputs the absolute object file 'sample.h20' in extended HEX format. The following comment is appended to the file header:

```
:This_is_sample
```

Example

```
tuconv -Fs24 -ra 0x0000,0x8000,,sample1.s24
-ra 0x14000,0x4000,+0x14000,sample2.s24 sample.abs
```

This command outputs 32K byte(-ra 0x0000, 0x8000) to an object file ' sample l.s24' of 24-bit Motorola S format from address 0x0000. Subsequently, it adds 16K byte(-ra 0x14000, 0x4000) and +0x14000 offset(+0x140000)(the address is 0x28000) from address 0x14000 and outputs that to an object file 'sample2.s24' of 24-bit Motorola S format. When the offset(+0x140000) is added, data address field(address field of branch instruction and CALL instruction) that is originally based on 0x14000 is not converted according to address 0x28000. It simply adds 0x14000 to the address field of the S-format.

Example

```
tuconv -Fs24 -l -rb sectionA,0x1000,,final.s24 sample.abs
```

This command outputs 4K byte(0x1000) to an object file 'final.s24' in 24-bit Motorola S format(-Fs24) from the start address of section A. In addition, the converting information such as output file mapping section is output to the console(-I).

Example

```
tuconv -Fs24 -rb sectionA,,,final.s24 sample.abs
```

This command outputs the whole section A to an object file 'final.s24' in 24-bit Motorola S format(-Fs24).

Example

```
tuconv -Fs24 -rb sectionA,0x3000,,final1.s24
-rb sectionB,0x1000,,final2.s24 sample.abs
```

This command outputs 12K byte(0x3000) to an object file 'final l.s24' in 24 bit Motorola S format (-Fs24) from the start address of section A. Subsequently, it outputs 4K byte(0x1000) to an object file 'final 2.s24' of 24 bit Motorola S format (-Fs24) from the start address of section B. If some section other than section A and B is included within the specified size, this command is not output. Similarly, the section with OVERAY attribute is not output.

Example

```
tuconv -P 0x8000,0x1000,0x00, sample.abs
```

This command outputs 4K byte(0x1000) to an object file 'sample.h16' in Intel HEX format from address 0x8000.(The output file name is omitted with -P.) The empty area with no object is filled with 0x00 in -P option.

Example

```
tuconv -Fs24 -lf cv.cnv -ra 0x0000,0x8000,+0x1000,sample1.s24
-rb sectionA,,0x0000,sample2.s24
-P 0x0000,0xffff,0x00,sample1.s24 sample.abs
```

This command adds 32K byte(0x8000) and +0x1000 offset and outputs to an object file 'sample l.s24' in 24-bit Motorola S format(-Fs24) from address 0x000. In -P option, the output range is 64K byte from 0x0000 to 0xffff. The empty area with no object is filled with 0x00 in -ra option. Subsequently, it outputs the start address 0x0000 of section A to 'sample2.s24'. The converting information such as mapping sections and padding area address information for the respective output files is output to list file 'cv.cnv.'(-If cv.cnv) At this time, the converting information is never output to the console.

Part 3 Option Details

Chapter 1 Rules for Specifying Options

Rules

- Specify options as alphabetical letters following a hyphen (-).
- A distinction is made between uppercase and lowercase letters.

```
cc870c -Nc1 -F0x00 -Ffilname.cmd filename.c  Incorrect
cc870c -Nc1 -F0x00 -ffilname.cmd filename.c  Correct
```

- When specifying two or more options, delimit the options with spaces.

```
cc870c -Nc1 -cD MACRO_NAME filname.c         Incorrect
cc870c -Nc1 -c -D MACRO_NAME filename.c      Correct
```

- When an option is specified with a string such as a filename, the option and string can be separated with a space, but the space is not mandatory.

```
cc870c -Nc1 -D MACRO_NAME filname.c          Correct
cc870c -Nc1 -DMACRO_NAME filename.c          Correct
```

- When a numerical value follows an option, do not separate the option from the numerical value with a space.

```
cc870c -Nc1 -O 2 filname.c                   Incorrect
cc870c -Nc1 -O2 filename.c                   Correct
```

- When a suboption follows an option, do not separate the option and suboption with a space.

```
cc870c -Nc1 -l s filname.c                   Incorrect
cc870c -Nc1 -ls filename.c                   Correct
```

- When specifying multiple suboptions, do not separate the main option from the suboptions or the suboptions from each other with spaces.

`cc870c -Nc1 -lsxw80 filname.mac`	s, x, and w are suboptions.
`cc870c -Nc1 -ls -lx -w80 filename.mac`	This is the same as the above.

- When a suboption takes an argument, delimit the suboption and argument with a space.

`cc870c -Nc1 -lffilename.lst filname.mac`	Incorrect
`cc870c -Nc1 -lf filename.lst filename.mac`	Correct

- When the same options are specified, the first option is effective.

`cc870c -Nc1 -lw180 -lw100 filname.mac`	-lw80 is effective

- Options between square brackets ([]) in the manual are optional.

`cc870c -Nc1 -w2 filname.mac`	Correct
`cc870c -Nc1 -w filename.mac`	Correct, -w is not necessarily the same as -w2.

Chapter 2 Conventions Used in Option Descriptions

Target tool

CC	ASM	LINK	LIB	CONV
x	x	*	-	x

Description

- The following shows the relationship between symbols and tools:

  CC CC Driver (CC870C, etc.)
  ASM Assembler (ASM870C, etc.)
  LINK Linker (TULINK,etc.)
  LIB Librarian (TULIB)
  CONV Object converter (TUCONV)

  See the Assembler Language Reference for details of Macro Preprocessor(TUMPP) option.

- The symbol under the target tool has the following meaning:

  x : Shows that the option is valid
  * : Shows the option can be specified, but that the function is valid for another tool, passed on to the appropriate tool.
  - : Shows that the option cannot be specified.

  The following section details each option and its parameters in alphabetical order.

Chapter 3 Details of Options

-# Display Path and Options

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-#
```

Function Displays the compiling or assembling process.

Description
- When the driver performs compiling and assembling, this option displays in sequence which tools are used, in what order, and with what options they were activated.
- Compiling and assembling are not actually performed.

Example
```
cc870c -Nc1 -# file1.c file2.c
```
With the 'cc870C file1.c file2.c' command line, it is possible to check which tool is activated with which option(s). Some options are automatically added by the driver. This option allows the user to check which options are added.

-A Compiles in accordance with ANSI specification

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-A

Function Compiles in accordance with ANSI specification.

Description

- In the default value of C compiler, the following two items do not apply correspondingly to ANSI specification.

  Integral promotion

  Arithmetic conversion for multiplication, division

  The -A option allows compiling in accordance with ANSI specification.
- It is recommended that the -A option is used to write a program which has an effective portability.

Example

```
cc870C –Nc1 -A file.c
```

-D Define Macro

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-D<definition name>=<definition>
-D<definition name>

Function Defines a preprocessor macro.

Description

- Defines a macro in the same way as using the preprocessor #define command. The two formats

```
-D<definition name>=<definition>
-D<definition name>
```

have the same result as entering the following at the beginning of a source file.

```
#define <definition name>=<definition>
# define <definition name>
```

- The C Compiler allows you to define up to 255 preprocessor macros by specifying the -D option repeatedly.
- In Assembler Preprocessor, you can define up to 20 preprocessor macros by specifying the -D option repeatedly.

Example

```
c870C -DBUFFER_LENGTH=256 -DDEBUG file1.c file2.c
```

This example has the same result as specifying the following at the beginning of a source file:

```
#define BUFFER_LENGTH=256
#define DEBUG
```

-E Preprocessor Output

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-E
```

Function Outputs the results of preprocessor processing of a source file to standard output.

Description
- When multiple files are specified, the results of processing of all files are output to standard output in the order in which the files were specified.

Example
```
cc870C –Nc1 -E file1.c file2.c
```
This command line executes only the preprocessor and outputs the results to standard output (-E).

-F Fill Value

Target tool

CC	ASM	LINK	LIB	CONV
*	-	x	-	-

Format

-F<value>

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

Description
- This option fills empty areas (padding areas) in an output section with the value specified in <value>.
- <value> must be specified as a 2-byte numerical value.
- The empty area is filled with the high then low bytes of the specified value from the padding start address.
- When the padding area is an odd number of bytes, the last byte is initialized with the high byte.
- Only memory areas with the I memory definition attribute in the command language file are padded. Memory areas without the I attribute (which specifies that an area can be initialized) are not padded.
- The padding value can be specified in link command file.

Example
```
tulink -F0xffff link.lcf
```
This command line links as specified in link.lcf and fills the empty areas in the output object file with 0xffff.

-F Select Object Format

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	-	x

Format

-F<object format>

Function Specifies the output format of the object converter.

Description

- The object formats are as follows:

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

- You cannot specify more than one object format at one time.
- If neither -Fh nor -Fs is specified, the command is processed assuming -Fh16 is specified. (When not specified = -Fh16)
- If a numeric value is omitted when specifying -Fh or -Fs, the command is processed assuming a value 16 is specified. (-Fh = -Fh16, -Fs = -Fs16)

Example

```
tuconv -Fh20 filename.abs
```

This command line creates an Intel extended HEX format file called 'filename.h20' from the absolute object file 'filename.abs'.

-I Directories for Input Files

Target tool

CC	ASM	LINK	LIB	CONV
x	x	-	-	-

Format

-I<path>

Function

Specifies the search path for include files and macro library files.

Description

- This option specifies the search path for include files specified in include instructions.
- <path> cannot be omitted.
- Do not specify the backslash (\) at the end of <path>.
- The C Compiler allows you to specify up to 31 directories.
- In Assembler Macroprocessor, and Assembler Preprocessor, you can specify up to 47 directories.
- Searches are performed according to the path specified in the -I option when the filename is specified with a relative path.
- When a file is specified with a relative path, the sequence in which it is searched for differs according to whether the file is specified as 'include<<filename>>' or 'include"<filename>"'. In the case of 'include<<filename>>', the search starts from (2), below:
  (1) The directory of the source files being.
  (2) The directory specified in the -I option. When the -I option is specified more than once, the INCLUDE directories of the specified directories are searched in the order specified.
  (3) When environmental variable THOMExxx has been set, the THOMExxx/include directory (where xx is the CPU)

Detail explanation `asm870c -I/usr/common -I../headers usr/a870C/filename.asm`

The following examples show how the files are searched for when the above is specified. The search stops immediately the target file is located. The environmental variables are set as below.

```
THOME870C=/commom/asmsys/tlcs870C
```

```
$include "def.h"
```

File 'def.h' is searched for as follows:

(1) usr/a870C/def.h (directory containing source file(s))
(2) /usr/common/def.h (/usr/common directory)
(3) ../header/def.h (../headers directory)
(4) /commom/asmsys/tlcs870C/include/def.h

```
$include <def.h>
```

File 'def.h' is searched for as follows:

(1) /usr/common/def.h (/usr/common directory)
(2) ../header/def.h (../headers directory)
(3) commom/asmsys/tlcs870C/include/def.h

```
$include "/usr/include/def.h"
```

Because the filename is specified with an absolute path, the directory remains unchanged even when the -I option is specified.

-J Select Kanji Mode (Japanese version only)

Target tool

CC	ASM	LINK	LIB	CONV
X	X	-	-	-

Format `-J`

Function Recognizes Kanji code.

Note Do not use this option for English version.

-L Directories for Input Files

Target tool

CC	ASM	LINK	LIB	CONV
*	-	x	-	-

Format

-L <path>

Function

Specifies the search path for input files for the linker.

Description

- The linker searches for object files, library files and command language files according to the path specified by this option.
- <path> cannot be omitted.
- Do not specify the backslash (\) at the end of <path>.
- 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.
- Files are searched for in the following order:
  (1) The current directory.
  (2) The directory specified in the -L option when the linker was activated. When the -L option is specified more than once, the specified directories are searched in the order specified.
  (3) When environmental variable THOMExxx has been set, the THOMExxx/lib directory (where xx is the CPU)

Detail explanation

```
tulink -L/usr/common -L../headers file1.obj usr/a870C/file2.obj file.lib
```

-Mi No Macro Processing

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-Mi

Function Controls the process so that Macro processor is not activated.

Description
- When not specified, Macro processor is activated.

Example
```
cc870C –Nc1 -Mi file1. mac
```
'file1.mac' is processed by Assembler preprocessor and assembler, but not by Macro processor.

-Nc Select Memory Style(TLCS-870/C,C1 only)

Target tool

CC	ASM	LINK	LIB	CONV
x	x	x	-	-

Format

-Nc<memory style>

Function Specifies CPU type or memory style of TLCS-870/C,C1.

Description
- Specify the Memory Style as a value of 0 to 3 in <memory style>.
- Be sure to specify this option. It becomes an error when this option is omitted.

Memory Style	Function
0	TLCS-870/C series
1	TLCS-870/C1 series Within 64Kbyte
2	TLCS-870/C1 series 96Kbyte
3	TLCS-870/C1 series 128Kbyte

Example
```
cc870C –Nc1 file1.c
```
file1.c is processed as TLCS-870/C1 within 64Kbyte by compiler, assembler and linker.

-O Select Optimization Level

Target tool

CC	ASM	LINK	LIB	CONV
x	x	-	-	-

Format

-O[<optimization level>][<optimization type>]

Function Specifies the optimization level of output code.

Description

- Specify the optimization level as a value in <optimization level>.
- When this option is omitted, the optimization level is as specified in the default value.
- The optimization levels for each tool are shown in the following tables.

C Compiler

Level	Function
0	Minimum optimization (default) Stack release absorption. Branch instruction optimization Deletion of unnecessary instructions
1	Basic block optimization Propagation of copying restricted ranges. Gathering of common partial expressions in restricted ranges.
2	Optimization of more than basic blocks Propagation of copying whole functions. Gathering of common partial expressions of whole functions
3	Maximum optimization Loop optimization and other miscellaneous optimization

Assembler

Level	Function
0	No optimization
1	Optimization (default)

Additional Note

- In case of C source file, to deterrent optimization for assembler, specify ”-Wa –O0”.

Example

```
cc870C –Nc1 -O3 file1.c file2.c
```

This command implements optimization to level 3 and compiles, assembles and links the files.

-P Run Preprocessor Only

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-P
```

Function Executes only preprocessor processing.

Description
- No compiling is performed.
- Only the preprocessor is run to create a file with the name of the source file and extension '.i'.

Example
```
cc870C –Nc1 -P file1.c file2.c
```
This option executes only the preprocessor and outputs the execution result to a file (-P).

-P Fill Value

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	-	x

Format

-P[<start address>],<size>,<value>,[<output file name>]

Function

Outputs <size> area to the file specified with <output file name> from <start address>. Unused area is initialized by the specified <value>.

Description

- When this option is used with “-ra” or “-rb” option, the address moved by “-ra” or “-rb” option is used as <start address>.
- <size> specifies 32-bit unsigned integer in byte. “K” following the numerical value represents kiro byte and “M” is mega byte.
- <value> is 1 byte integer.
- <start address> specifies 32-bit unsigned integer. When <start address> is omitted, the start address is address 0.
- This option can be specified multiple times.

Example

```
tuconv -ra 0x0,0x8000,+0x1000, -P 0x1000,32k,0x00, file.abs
```

Moves 32K byte(0x8000) to address +0x1000 from address 0x0000 of file.abs., embeds 0x00 to the empty area and outputs to “file.h16”.

-S Create Assembly Source

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-S

Function

Activates the compiler or macro processor and preprocessor to create an assembler source file.

Description

- No assembling or linking is performed.

Examples

```
cc870C –Nc1 -S file1.c file2.mac
```

This command compiles 'file1.c' to output 'file1.asm' (-S). File 'file2.mac' is processed by Macro processor and assembly preprocessor to output 'file2.asm' (-S). No processing is performed after assembling.

-T Set Environment Variable

Target tool

CC	ASM	LINK	LIB	CONV
-	-	X	X	X

Format

-T<CPU code>

Function Sets the environment variable searched by the tools.

Description

- <CPU code> is as follows:

Series	TLCS-870/X	TLCS-870	TLCS-870/C,C1
CPU Code	870X	870	870C

- Each tool uses <CPU code> to search for the environment variable THOME<CPU code>. If a path is set in that environment variable, it is recognized as the standard directory.

Additional Note

- When using drivers, this option is automatically set for Macro processor, preprocessor, and linker.

Example

```
cc870c –Nc1 -T870C -I/usr/commo file1.mac
```

This command adds the directory specified in THOME870C,C1 (-T870C) as well as the directories (standard) specified in environment variable THOMExxx.

-U Undefine Macro

Target tool

CC	ASM	LINK	LIB	CONV
X	-	-	-	-

Format

-U<definition name>

Function Invalidates the definition of the preprocessor macro in <definition name>.

Description

- This option has the same result as specifying #undef<definition name> at the start of a source file.
  The following format:

```
-U<definition name>
```

  has the same result as when the following is specified at the start of the source file:

```
#undef <definition name>
```

- The -U option is valid, regard of the order of the arguments, when the -D option is used as an argument on the command line to define a <definition name> with the same name.
  For example, the following is the same as when -DDEBUG is not specified:

```
cc870c –Nc1 -DDEBUG -UDEBUG filename.mac
```

- The C compiler allows you to invalidate up to 255 preprocessor macros by specifying the -U option repeatedly.
- In Assembler Preprocessor, you can invalidate up to 20 preprocessor macros by specifying the -U option repeatedly.

Example `cc870c –Nc1 -UDEBUG file1.c file2.c`

-V Output Version Number

Target tool

CC	ASM	LINK	LIB	CONV
x	x	x	x	x

Format

```
-V
```

Function Outputs Assembler version number to console.

Description
- When Assembler is activated, startup messages such as Assembler version number are output to console.

Example
```
cc870C –Nc1 -V file1.c file2.c
```

-W Pass Options to Tool

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-W<tool>,<option>[,<option>...]
```

Function Passes one or more options to the specified tool

Description
- This option passes the option(s) specified in <option> as arguments when activating the tool specified in <tool name>.
- The tool is specified in <tool name> using one of the following letters.

Tool name	Tool
p or 0	Parser (for C compiler)
2	Code generator (for C compiler)
m	Macro preprocessor
a	Assembler
l	Linker

- Multiple options can be specified by delimiting <option> with commas.

Example
```
cc870C –Nc1 -Wa,-o,boo.asm foo.mac
```

This command line passes the two options '-o' and 'boo.asm' to MacroPreprocessor.

-XE Ignore Escape Sequence

Target tool

CC	ASM	LINK	LIB	CONV
*	x	-	-	-

Format

```
-XE
```

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

Description
- This option does not need to be specified when a driver is used to run each of the tools including Assembler.

Example
```
asm870C -J -XE file1.asm
```

-XF No Deletion of Assembler Source

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-XF
```

Function Driver does not delete the assembler source files that are intermediate results of the process.

Description
- Compiles, assembles, and links files without deleting the assembler source files that are intermediate results of the process.

Example
```
cc870c –Nc1 -XF file1.c
```
'file1.c' is compiled to output 'file1.asm' (-XF). The file is assembled and linked.

-XS Code Size Optimization

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-XS
```

Function Specifies the output of minimum object code size.

Description
- When this option is specified, part of optimization is skipped.

Example
```
cc870c –Nc1 -XS file1.c file2.c
```
This command specifies the output of minimum object code to improve memory utilization. However, execution speed may deteriorate.

-Xec Change Type of Enumeration Constant

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-Xec

Function

Determine the type of enumeration constant according to the scope of enumerator values.

Description

- According to the scope of enumerator values, this option determines the type of enumeration constant to be one of *unsigned char*, *signed char*, *unsigned int*, or *signed int*.
- The type of enumeration constant in order of the following.

Scope of Enumerator Values	Type of Enumeration Constant
0 to 255	unsigned char
-128 to 127	signed char
0 to 32767	unsigned int
-32768 to 32767	signed int

- For enumeration constants outside the above scopes, Compiler outputs Error-343: Out of range for enum constant.

Supplement

- Supplement Because this option is not ANSI-compliant, it cannot be used in combination with the -A option. Neglect of this limitation results in generation of warning

  Options -Ao and Am can be used in combination.
- Specifying this option helps to increase memory efficiency.

Example

cc870c –Nc1 –Xec file.c

```
/* unsigned char */
  enum BEST1{uc_mini=0,uc_max=255};
/* signed char */
  enum BEST2{sc_mini=-128,sc_max=127};
/* unsigned int */
  enum BEST3{ui_mini=0,ui_max=32767};
/* signed int */
  enum BEST4{si_mini=-32768,si_max=32767};
  enum BEST1    i=uc_mini;/* i is unsigned char*/
  enum BEST2    j=sc_mini;/* j is signed char */
  enum BEST3    k=ui_mini;/* k is unsigned int */
  enum BEST4    l=si_mini;/* l is signed int */
```

-Xi Run Macro Processor

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-Xi
```

Function Activates Macro processor before Assembler when creating object files from C language source files.

Description
- Normally, only Assembler is activated when creating object files from C language source files.

Example
```
cc870c –Nc1 -Xi file1.c file2.c
```
Normally assembler source files are created by the compiler and these files are then assembled. However, this command inserts macro processor processing between compiling and assembling.

-Xns Integration of Stack Freeing

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-Xns
```

Function Suppress the optimization to integrate freeing of the stack.

Description
- This option suppresses the optimization to integrate freeing of the stack.

Supplement
- Specifying this option helps to increase RAM efficiency. However, code efficiency slightly decreases and the execution speed slows down.

-Xr Set Default Function Modifier (__adecl)

Target tool

CC	ASM	LINK	LIB	CONV
X	-	-	-	-

Format

```
-Xr
```

Function Specifies '__adecl' as the default function modifier.

Description
- Compiling is performed as if '#pragma adecl' is specified at the beginning of the source file.
- When the standard library function is used, note that this option can not be specified.

Example
```
cc870c -Nc1 -Xr file1.c file2.c
```

-Xub Select Unsigned Characters

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-Xub

Function

Specifies that bit field members declared without 'signed' or 'unsigned' are processed as unsigned.

Description

- When this option is not specified, bit field members declared without 'signed' or 'unsigned' are processed as signed.

Example

```
cc870c –Nc1 –Xub filename.c
```

When built into filename.c

```
struct code {int seg:4;  int off:12;};
```

is equivalent to declaring the following:

```
struct code {unsigned int seg:4; unsigned int off:12;};
```

-Xuc Select Unsigned Characters

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-Xuc

Function

Handles the char type declared without adding signed or unsigned as unsigned char and strings as unsigned char.

Description

- When this option is not specified, char declared without 'signed' or 'unsigned' are processed as signed.

Example

```
cc870c –Nc1 -Xuc filename.c
```

When built into filename.c

```
char message[]="Hellow World!";
```

is equivalent to declaring the following:

```
unsigned char message[]="Hellow World!";
```

-Xw Change bit field assignment order

Target tool

CC	ASM	LINK	LIB	CONV
X	-	-	-	-

Format

```
-Xw
```

Function Changes bit field assignment order.

Description
- The members of the bit field are assigned from the most significant bit(MSB). When this option is specified, they are assigned from the least significant bit.

Additional Note
- See the Programmer's Guide file for details.

Example
```
cc870c –Nc1 -Xw file.c
```

```
Bit field definition in file.c.
```

```
struct field1 {
  unsigned char a:1;
  unsigned char b:2;
  unsigned char c:3;
  unsigned char d:1;
  unsigned char e:3;
  unsigned char f:2;
}
```

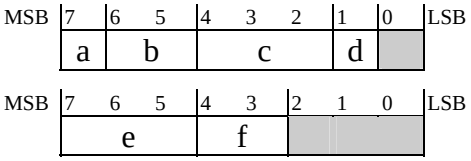


Figure 3.2 Assignment image of default field 1
(shows empty bit)

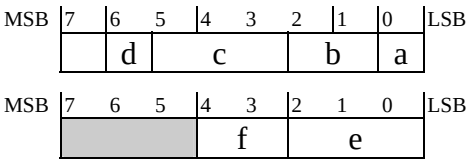


Figure 3.2 When specified -Xw, assignment
of field 1(shows empty bit)

-ZA -ZC -ZD -ZT Set Default Section Size

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-ZA<size attribute>
-ZC<size attribute>
-ZD<size attribute>
-ZT<size attribute>

Function Specifies the default section size attribute.

Description
- Specify 'tiny', 'near', or 'far' in <size attribute>.
- The following table shows the relationship between this option and the section's default <size attribute>.

		Default		
	Section	870/X	870	870/C,C1
-ZA	area section	far	near	near
-ZC	const section	far	near	near
-ZD	data section	far	near	near
-ZT	code section	far	near	near

Additional Note
- This option has similar function to -Za, -Zc, -Zd, -Zi, and -Zt. In contrast to -Za, etc., this option cannot be used to change the name of a default section.

Example
```
cc870c –Nc1 -ZA near file1.c file2.c
```
This command compiles source files file1.c and file2.c using 'near' as the size attribute of any area sections in those source files.

-Za -Zc -Zd -Zi -Zt Set Default Section Name

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-Za<section name>[,<size attribute>]
-Zc<section name>[,<size attribute>]
-Zd<section name>[,<size attribute>]
-Zi<section name>
-Zt<section name>[,<size attribute>]

Function Specifies the default section name.

Description
- The default section name is used when this option is not specified.
- Specify <section name> as a character string of maximum 32 characters.
- Specify 'tiny', 'near', or 'far' in <size attribute>.
- The following table shows the relationship between this option and the section's default <section name>.

	Default		
	870/X	870	870/C,C1
-Za	f_area	n_area	n_area
-Zc	f_const	n_const	n_const
-Zd	f_data	n_data	n_data
-Zi	io_XXX	io_XXX	io_XXX
-Zt	f_code	n_code	n_code

Example
```
cc870c –Nc1 -Za newarea,near file1.c file2.c
```

-c Create Object but not Link

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-c
```

Function Creates a relocatable object file.

Description
- Compiling and assembling, or just assembling, are performed and a relocatable object file is created. The linker is not activated.
- When multiple source files are specified as command line arguments, each file is compiled and assembled to create a relocatable object file.

Example
```
cc870c –Nc1 -c file1.c file2.c
```

-c Specify a comment

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	-	x

Format

```
-cc<comment>
-cf<filename>
```

Function Specifies a comment to be inserted into an object file output by the object converter.

Description
- <comment> is a character string. <comment> cannot include spaces.
- Comments in the file specified in <filename> are recognized as one comment per line. Multi-line comments can be written in the file.
- When converting a file to the Intel object format, the colon cannot be used as the first character of a comment. If specified, it is ignored.

Example
```
tuconv -cc this_is_comment file.abs
```
Creates an object file in Intel HEX format and inserts the comment 'this_is_comment'.
```
tuconv -Fs24 -cf comment.cmt file.abs
```
Creates an object file in 24-bit Motorola S format and creates comments from the character strings in the file 'comment.cmt'.

-d Delete a Module

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	x	-

Format

-d[v] <library filename><module list>

Function Deletes one or more modules from a library file.

Description
- This option deletes the module(s) specified in <module list> from the library file specified in <library file>.
- Specify the modules in the module list delimited with spaces.
- The 'v' sub-option outputs the names of the deleted modules to standard output.

Example
```
tulib -dv libfile.lib module1 module2
```
This command deletes modules 'module1' and 'module2' from library file 'libfile.lib'.

-e Create Error List File

Target tool

CC	ASM	LINK	LIB	CONV
*	x	x	x	x

Format

-e<filename>

Function Outputs all error messages to one file.

Description
- This option outputs all errors that occur during program execution to the file specified in <filename>.
- Warning error messages are also output to the error list file.
- When a driver is used, all error messages from all tools activated by the driver are output to the specified file.
- When a fatal error occurs, processing ends immediately and no error list file may be created.

Additional Note
- Tools other than drivers add error messages to any files previously specified with this option, if they exist.

Example
```
cc870c –Nc1 -e errlst.err file1.c file2.c
```

-f Read Option from a File

Target tool

CC	ASM	LINK	LIB	CONV
x	x	-	x	x

Format

-f <filename>

Function

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

Description

- The options specified on the command line can be written in a text file and specified in <filename>. The list of options is then automatically read in from that file.
- Command parameters can be specified across multiple lines in an option list file.

Example

```
cc870c –Nc1 -ffilename.cmd
```

In this example, the cc870c reads the command parameters from command file 'filemane.cmd'. The following is an example of the content of 'filename.cmd'.

```
-O2 -DMACRO
-ooutfile.abs
file1.c
file2.asm
file3.rel
file4.lib
```

-g Create Debugger Information at assemble phase

Target tool

CC	ASM	LINK	LIB	CONV
x	x	-	-	-

Format

-g[<level>]

Function

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

Description

- This option outputs source level debugging information or symbolic debugging information to an object file.
- If the source file is written in C, the source level debugging information is output in C. Likewise, if the source file is in assembly language, the debugging information is in assembly.
- You can select "source level debugging infomation" or "symbolic debugging infomation" by designate <level>. Option "-g" and "-g0" are the same.

Level	Function
0	Outputs source level debugging infomation
1	Outputs symbolic debugging infomation

- When this option is omitted, no source level debugging information is output to the object file.

Additional Notes

- No source level debugging information is output for the macro processor or assembler preprocessor. Therefore, carry out source level debugging on Assembler source resulting from macro expansion in the case of macro processor and assembler preprocessor source programs.

Example

```
cc870c –Nc1 -g file1.c file2.c
```

-g Create Debugger Information at linking phase

Target tool

CC	ASM	LINK	LIB	CONV
*	-	x	-	-

Format

```
-ga
-gm <filename>
```

Function Outputs debugging information to an ABS file.

Description
- This option outputs debugging information stored in REL file which is specified by a linkage editor's input file, into the ABS file.

sub option
- Sub option "-ga" or "-gm" must be specified to output the debugging information into the ABS file.

-ga
- The debugging information of every REL file is stored into the ABS file.

-gm <filename>
- The debugging informations of the REL file which is specified by a <file name> is stored into the ABS file.
- This option can be specified as many times as desired.

Example

```
tulink -ga file1.rel file2.rel
tulink -gm file1.rel file1.rel file2.rel
```

-l Create List File

Target tool

CC	ASM	LINK	LIB	CONV
*	x	x	-	x

Format

-l[<suboption>]

Function

Creates a list file.

Description

- Unless the 'f' suboption is specified, the list file takes the name of the source file plus an extension which is the preset value for the tool.

Tool	Extension
Assembler	.lst
Linker	.map
Macropreprocessor	.med

- The information output to the list file can be controlled according to the one-letter suboption specified after '-l'. Table shows the suboptions.
- Multiple suboptions can be specified after '-l'. However, suboptions (f and x) of the argument must be specified last.
- The following describes types of suboption and tools that can be used.

Suboptions

	ASM	LINK	CONV
No specification	x	x	x
a	-	x	-
f<filename>	x	x	x
x[<value>]	x	x	-

Suboption

In addition to controlling the information that is output, this option also controls the format of the list. The following details the operations specified by the suboption.

No specification

Only '-l' is specified. In this case, a basic list file is output, including a symbol list but no page break codes. The list file name takes the name of the source file plus an extension which is the preset value for the respective tools.

- When “-I” is specified in tuconv, the converting information is output to the console. If “-I” is not specified, the information is never output. Thus only object conversion is executed.

a

All link information is output to the link information list file.

f<filename>

The list file is output with the name specified in <filename>, which is mandatory.

x This suboption outputs a cross reference to the list file.

- In the case of the linker, a cross reference of wide- area symbols is output to the wide-area symbol list in the list file in addition to basic link information. When no 'g' suboption is specified, it is automatically assumed and the public symbol list output at the same time.

Example `cc870c –Nc1 -la file.c`

This command creates Linker list file 'file.map'.

-l Output Module Symbol Information

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	x	-

Format

-l <library filename>[<module lis>]

Function Outputs symbol information for modules in a library file.

Description
- This option outputs symbol information for modules in a library file.
- Specify the target library file in <library filename> and the modules in that library file for which symbol information is to be output in <module list>. The following is output as information on symbols in a module:
  - Module name
  - Size
  - External definition symbols
  - External reference symbols
- Specify the names of modules in <module list> delimited with spaces.
- When no <module list> is specified, information is output for all modules in the specified library file.
- An error occurs if the modules specified in <module list> do not exist in the specified library file.

Example `tulib -l libfile.lib module1 module2`

Symbol information for modules 'module1' and 'module2' in library file 'libfile.lib' is output in the following format:

```
MODULE INFORMATION :
Name                                    Size Type
module1                                 279  relocatable
module2                                 353  relocatable
PUBLIC SYMBOL(S) :
module1                             init
module2                             table
EXTERN SYMBOL(S) :
modume1                             No Symbol
modume2                             calc
                                    init
```

-m Select Memory Model(TLCS-870/X only)

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

-m<memory model>

Function

Sets TLCS-870/X memory model.

Description

❑ <memory model> is as follows:

Memory model	Description
-ms	Compile in small model
-mm	Compile in medium model
-ml	Compile in large model

❑ The default when this option is omitted is -ms.

Example

```
cc870X -ms file.c
```

This command compiles 'file.c' in small mode.

-o Set Output Filename

Target tool

CC	ASM	LINK	LIB	CONV
x	x	x	-	x

Format

-o<filename>

Function

Specifies the filename of the final output file.

Description

- The final output file is created with the name specified in <filename>.
- <filename> is mandatory.
- When the tools are activated by a driver, the last- activated output file is specified.
- The last output file of each tool is as follows. When this option is omitted, the final output file takes the name of the source file plus the appropriate extension.

Tool	Final output file	Extension
MPP	Macro preprocessor output file	.asm
ASM	Relocatable object file	.rel
LINK	Absolute object file	.abs
TUCONV	Converted object file	Note

Note : Extension is depend on the object format.

- In the case of CC driver, the final output file differs according to the specified options:

Option	Final output file
-c	Relocatable object file (last tool is ASM)
-P	Macropreprocessor output file
-S	Assembler source file

- When this option is specified by a driver, a filename specified with the '-o' option is ignored when two or more files are created.
- This option must be specified when creating a relocatable object file by specifying '-r' with the linker.

Examples

```
cc870c –Nc1 -o test.abs file.c
```

This command activates CC driver to compile C source file 'file.c' and create the absolute object file 'test.abs'.

-r Select Incremental Linking

Target tool

CC	ASM	LINK	LIB	CONV
*	-	x	-	-

Format

```
-r
```

Function Creates a relocatable object file (incremental linking).

Description
- This option is used when relinking an object file that has already been linked and a relocatable object file that has not yet been linked.
- When linking relocatable object files, this option specifies that the result of linking is output not as an absolute object file but as a relocatable object file.
- When specifying this option, also specify the name of the output file using the '-o' option. An error occurs if the output filename is not specified.
- The input sections are linked according to the section instructions in the command language. The mapping address is not, however, defined. Output files linked using this options are therefore relocatable object files.
- When this option is specified, a warning message is output and the memory definition section is ignored if one exists in the command language file.

Example
```
cc870c –Nc1 -o file.rel -r file1.c file2.c
tulink file.rel file3.rel
```
This command creates a relocatable object file named 'file.rel'. 'file.rel' and 'file3.rel' are then linked to create 'file.abs'.

-r Replace Modules

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	x	-

Format

-r[cuvw] <library filename><module list>

Function

Creates a library file and records and updates modules.

Description

- This option records and updates the modules specified in <module list> in the library file specified in <library filename>.
- There are three ways to specify the modules in <module list>. These three ways can be combined by delimiting them with spaces.
  - To specify the filename of the object file when specifying modules in an object file.
  - Specify only the library filename when all modules in a library file are being indicated.
- If a module specified in <module list> does not exist in the specified library file, it is inserted in that library file.
- If a module specified in <module list> does exist in the specified library file, the module with that name in the library file is updated with the module specified.
- If the specified library does not exist, a message is output and a new file is created. The modules are then recorded in the new library.

Suboptions

c
- Suppresses message output when creating a library file.

u
- Updates a module in a library file only when the module specified in <module list> has a more recent date than the module in the library file.
- Updates only those modules specified in <module list> that have more recent dates than those in the library file.

v
- Outputs the name of the recorded or updated module to standard output (normally console).
- Returns an error when a module specified in <module list> does not exist.

w
- When the module specified in <module list> is dated more recent than those in the library file, it is updated.
- If the module specified in <module list> does not exist in the library file, it is registered to the library.

Example `tulib -r libfile.lib file.rel`

This command records or updates module 'file.rel' in library file 'libfile.lib'. The module is recorded in the library file if it does not already exist, or is updated if it does already exist.

-ra Object Output Range by address specification

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	-	x

Format

-ra[<block start address>],[<size>],

[<offset or moved start address>],[<Output file name>]

Function Selects <size> object from <block start address> of input file and outputs the selected object to the file specified by <output file name> after the address is moved with <offset or moved start address>.

Description
- <block start address> specifies 32-bit unsigned integer.
- When <block start address> is omitted, the start address is regarded as address 0.
- <size> specifies 32-bit unsigned integer.
- When <size> is omitted, all objects after <start address> is output.
- When the first mapping address of the object differs from the address at execution, <offset or moved start address> is used to set the first mapping address. Additionally, it is used to move the address once at writing the object to EPROM, etc.
- The object code(address parts in the code) is not changed.
- <offset or moved start address> is specified as follows:
  <integer value> : Specifies the moved start address.
  +<integer value> : Adds the offset to the address.
  -<integer value> : Subtracts the offset to the address.
- <output file name> specifies the object output file name.
- When <output file > is omitted, the file name is changed to that ' h16', '.s24', etc. add to the same as the input file name.
- When this option is specified multiple times and <output file name> is the same as that of the multiple options, the respective options are collected into one output file.

Example
```
tuconv -ra 0x12000,0x1000,,file.h20 file.abs
```
Outputs 0x1000(4K byte) in Intel extended HEX format from start address 0x12000.

-rb Object Output Range by section specification

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	-	x

Format

-rb<section name>,[<size>],

[<offset or moved start address>],[<Output file name>]

Function

Selects <size> object specified with <section name> from input file and outputs the selected object to the file specified by <output file name> after the address is moved with <offset or moved start address>.

Description

- <section name> specifies output section name.
- <section name> is the section name determined at linking. The output section name specified at linking is the section name.
- <size> specifies 32-bit unsigned integer.
- When <size> is omitted or over the specified section size , only the specified section is output.
- When the first mapping address of the object differs from the address at execution, <offset or moved start address> is used to set the first mapping address. Additionally, it is used to move the address once at writing the object to EPROM, etc.
- The object code(address parts in the code) is not changed.
- <offset or moved start address> is specified as follows:

  <integer value> : Specifies the moved start address.

  +<integer value> : Adds the offset to the address.

  -<integer value> : Subtracts the offset to the address.
- <output file name> specifies the object output file name.
- When <output file name> is omitted, the output file name is the file name which 'h16', '.s24', etc. follow the same as the input file name.
- When this option is specified multiple times and <output file name> is the same as that of the multiple options, the respective options are collected into one output file.

Example

```
tuconv -Fh20 -rb sectionA,0x1000,,file.h20 file.abs
```

Outputs 0x1000(4K byte) in Intel extended HEX format from the beginning of section A.

-s Define SET Symbol

Target tool

CC	ASM	LINK	LIB	CONV
*	-	-	-	-

Format

-s <SET symbol>=<value>

Function Defines a symbol of Macro Preprocessor SET function.

Description
- Specify <value> as a 32-bit signed integer.

Example
```
cc870c –Nc1 –sAAA=1 file.mac
```

-t Output Module List

Target tool

CC	ASM	LINK	LIB	CONV
-	-	-	X	-

Format

-t[v] <library filename>[<module list>]

Function Outputs a list of modules in a library file.

Description
- This option outputs a list of the modules specified in <module list> in the library file specified in <library filename>. The list is output to standard output (normally console).
- The module list contains the information which Module name, size in bytes, attribute, creation date and time.
- Specify modules in <module list> delimited by spaces.
- The 'v' suboption outputs size and creation date in addition to the module name.
- When <module list> is omitted, a list of all modules in the library file is output.

Example
```
tulib -tv libfile.lib
```
This command lists information on all modules in library file 'libfile.lib'. An example of output is shown below. From left to right, each line includes the module name, size, attribute, and creation date.
```
module1  279  relocatable        Mar 07 22:05 1992
module2  353  relocatable        Mar 07 22:43 1992
```

-u Delete All Predefined Macros

Target tool

CC	ASM	LINK	LIB	CONV
x	-	-	-	-

Format

```
-u
```

Function Invalidates all predefined macros.

Description
- The following table lists predefined macros.

Predefined macro	Remarks
__LINE__	Common
__FILE__	Common
__DATE__	Common
__TIME__	Common

Example
```
cc870c –Nc1 -u file1.c file2.c
```

-u Record Undefined Symbol

Target tool

CC	ASM	LINK	LIB	CONV
*	-	x	-	-

Format

```
-u<symbol>
```

Function Records an undefined symbol in the symbol table.

Description
- This option records <symbol> as an undefined symbol in the symbol processing table.
- The symbol processing table is initially empty and is used for referencing unresolved symbols in order to forcibly load a routine. It is useful to, for example, load modules only from a library.

Example
```
tulink -u symbol link.lcf
```

-w Select Warning Level

Target tool

CC	ASM	LINK	LIB	CONV
*	x	x	-	-

Format

```
-w[<warning level>]
```

Function Specifies the warning level.

Description
- Specify the warning level as a number in <warning level>.
- The default value when this option is not specified is '-w1'.
- Note that specifying '-w' without a number is equivalent to '-w0', and no warnings are output.
- See the section on error messages for the warnings that are output at each level.
- The warning level range of each tool is as follows:
  - C Compiler 0-3
  - Assembler 0-1
  - Linker 0-1
- The warnings are output for all levels at and below the specified level.

Example
```
cc870c -Nc1 -w2 file1.c file2.c
```
This command outputs messages for warning levels 1 and 2 in addition to normal error messages.

Part 4 Formats

Chapter 1 Assembler List Format

1.1 Assemble List

The format of the assemble list is as follows:

```
Location  Object                    Ins   Line Source Statement
xxxxxxxx  yyyy yyyy yyyy yyyy  r +i nnnn  mmmmm sssss...
          yy [zzz]
  vvvvvvvv
```

x..x : Location counter (hexadecimal)

v..v : Value set by EQU directive (4-digit hexadecimal)

y..y : Object code
Normally left aligned.

z..z : yy indicates one code when using the DFt directive, and zzz shows the number of codes.

r : "R" is displayed when a relocatable term is included.

+i : Shows "include" nesting level (one decimal digit)

n..n : Line number
This is the sequential number of each file if more than one include file exists.

m..m : Line number
This is the sequential number of all files including include files.

s..s : Source code

1.2 Symbol List Format

The following shows the format of symbol lists.

```
Symbol           Category Value     Attribute   Cross_reference
ssssssssssssssss c z aaa  xxxxxxxx  r yyyyyyyy  dddd    kkkkk   kkkkk#
sssssssssssssssssssssssssssssssss
                 c z aaa  xxxxxxxx  r yyyyyyyy  dddd    kkkkk   kkkkk#
```

s..s : Symbol (Displayed on one line if 16 or fewer characters)

c : Section attribute

D : Data C : Code R : Romdata Empty : Other

z : Size

S : Small M : Medium L : Large Empty : Other

a..a : Class attribute

LAB : Label VAR : Variable

NUM : EQU value MOD : Module name

SEC : Section name

x..x : Symbol value

Class attribute = LAB, VAR: Address value

Class attribute = NUM : Definition value

Class attribute = SEC : Starting address value

Class attribute = MOD : Empty

Definition symbol : Undef

r : Relocatable attribute

R : Relocatable value A : Absolute value

y..y : Symbol scope

PUB : Public EXT : External Empty : Local

Class attribute = SEC : Section size

Class attribute = MOD: Not displayed

d..d : Section name

(Only when the section to which a label or a variable belongs is relocatable.)

k..k : Cross-reference. A pound sign (#) is added to the definition line. (The line numbers are sequential.)

```
Copyright(C) 1992 TOSHIBA CORPORATION All rights reserved
Tue Mar 30 11:02:58 1993

TLCS-870/C Relocatable Assembler V1.0a       [Page     1]  sam_asm1.lst
Runtime option : K:\BIN\ASM870C.EXE sam_asm1.asm

Location  Object            Ins   Line Source Statement

                          +0    1      1 -lw120
                          +0    2      2 $MAXIMUM
                          +0    3      3
  0000FEDC                +0    4      4 D16     EQU     0xfedc
  00ABCDEF                +0    5      5 D24     EQU     0xabcdef
                          +0    6      6
                          +0    7      7         public  VAR1,LAB1,LAB2
                          +0    8      8         extern  medium  ext_m1,ext_m2
                          +0    9      9
                          +0   10     10
                          +0   11     11 DT_1    section data    small
00000000                  +0   12     12 VAR1    dsb     1
00000001                  +0   13     13 VAR2    dsw     1
00000003                  +0   14     14 VAR3    dsl     1
00000007                  +0   15     15         dsb     1
                          +0   16     16 ;
                          +0   17     17 ;
                          +0   18     18 CD_1    section code    abs=0x8000
00008000  ECC8EFCDAB00    +0   19     19 LAB1:   ADD     XIX,D24
00008006  EDC8EFCDAB00    +0   20     20         ADD     XIY,D24
0000800C  D8C8EFCD        +0   21     21         ADD     WA,D24
D:\TEMP\sam_asm1.asm 21 : ASM870C-Warning-501 : Operand value is out of range
00008010  300000      R   +0   22     22         LD      WA,VAR1
00008013  310000      R   +0   23     23         LD      BC,ext_m1
                          +0   24     24 ;
                          +0   25     25 ;
                          +0   26     26 CD_2    section code    medium  align=2,2
00000000  00              +0   27     27         nop
00000300                  +0   28     28         org     0x300
00000300  00              +0   29     29 LAB2:   nop
00000301                  +0   30     30         align   0x10
00000310  00              +0   31     31         nop
                          +0   32     32 ;
                          +0   33     33 ;
                          +0   34     34 RD_1    section romdata large
00000000  01              +0   35     35 VAR4_abcdefghijklmno    db      1
00000001  0100            +0   36     36 VAR5    dw      1
00000003  01000000        +0   37     37 VAR6    dl      1
                          +0   38     38 ;
                          +0   39     39 ;
                          +0   40     40         END
D:\TEMP\sam_asm1.asm 40 : ASM870C-Warning-513 : "1" ignored extern symbol(s)

Assembly complete,     2 warning error(s)
```

```
TLCS-870C Relocatable Assembler V1.0a       [Page     2]  sam_asm1.lst

Symbol table listing
--------------------

Symbol           Category Value     Attribute

CD_1             C   SEC  00008000  A 16
CD_2             C M SEC  00000000  R 311
D16                  NUM  0000FEDC  A
D24                  NUM  00ABCDEF  A
DT_1             D S SEC  00000000  R 8
LAB1             C   LAB  00008000  A PUB       CD_1
LAB2             C M LAB  00000300  R PUB       CD_2
RD_1             R L SEC  00000000  R 7
VAR1             D S VAL  00000000  R PUB       DT_1
VAR2             D S LAB  00000001  R           DT_1
VAR3             D S LAB  00000003  R           DT_1
VAR4_abcdefghijklmno
                 R L LAB  00000000  R           RD_1
VAR5             R L LAB  00000001  R           RD_1
VAR6             R L LAB  00000003  R           RD_1
ext_m1                    00000000  R EXT
sam_asm1             MOD


Define    16 user symbol(s)
```

Chapter 2 Linker List Format

Information is output to the link list in the following format:

(1) Command file display (command file name and commands)
(2) Encountered error messages (in order encountered)
(3) Input module list (input file names and module names)
(4) Link map

The link map contains 7 items, each on one line.

Memory	:	Memory name
Out-sec	:	Output section name
Attri	:	Section attribute
Base	:	Input section starting address
Length	:	Input section size
In-sec	:	Input section name
(In-file)	:	Name of input file containing input section
Information	:	Other information

NORMAL	:	Normal section(Relocatable)
NORMAL:A	:	Normal section(Absorute)
Gap	:	Empty
Padding	:	Padding area
DUMMY	:	DUMMY section
NOLOAD	:	NOLOAD section
COPY	:	COPY section
OVERLAY	:	OVERLAY section

(5) Duplicate-definition public symbols (symbol name and defined file name)
(6) Unresolved external symbols (symbol name and reference file name)
(7) Symbol list

The symbol list is displayed as four items on individual lines:

Symbol	:	Name of public or local symbol
Address	:	Value (address) of symbol
In-sec	:	Name of input section
Cross-reference	:	Name of called input module

```
Toshiba Unified Linkage Editor V1.0a          [Page     1]     sam_asm1.map
Runtime option : K:\BIN\TULINK.EXE sam.lcf

  Command file : sam.lcf
    -la sam_asm1.rel sam_asm2.rel
    memory
    {
     data.s    : origin=0x0000 , length=0x80
     data.m    : origin=0x080 , length=0x0400
     code.m    : origin=0x8000 , length=0x1000
     romdata.m : org=0x9000  , length=0x1000
     data : org=0x10000   , len=0x10000
     code : org=0x20000   , len=0x10000
    }

  TULINK-Warning-511: Unresolved external symbol "EXT_L"
  TULINK-Error-209: Reference made to unresolved external symbol "EXT_L"

  Input files (modules)
    sam_asm1.rel (sam_asm1)
    sam_asm2.rel (sam_asm2)

  Link map
    Memory  Out-sec  Attri Base    Length     In-sec(In-file)        Information
  --------- -------- ----- ------- ---------- ---------------------- ----------
    data.s   DT_1   DATA         0          8 DT_1    (sam_asm1.rel)     NORMAL
    data.s   RD_1   ROMDATA      8          7 RD_1    (sam_asm1.rel)     NORMAL
    data.s                       F         71                           *** Gap ***
    data.m                      80        400                           *** Gap ***
    code.m   CD_1   CODE      8000         16 CD_1    (sam_asm1.rel)     NORMAL
    code.m   CD_2   CODE      8016        311 CD_2    (sam_asm1.rel)     NORMAL
    code.m                    8327        1D9                           *** Gap ***
    code.m   CD_A   CODE      8500          E CD_A    (sam_asm2.rel)     NORMAL
    code.m                    850E        AF2                           *** Gap ***
    romdata.m                 9000       1000                           *** Gap ***
    data                     10000      10000                           *** Gap ***
    code                     20000      10000                           *** Gap ***

  Unresolved external symbols
    EXT_L        : sam_asm2.rel

  Symbol table for sam_asm1.abs
      Symbol       Address  In-sec   Cross-reference
      ------------ -------- -------- ------------------------------------
    Input module : sam_asm1
      LAB1             8000 CD_1     sam_asm2
      LAB2             8316 CD_2     sam_asm2
      VAR1                0 DT_1     sam_asm2
      VAR2                1 DT_1
      VAR3                3 DT_1
      VAR4_abcdefghijklmno
                          8 RD_1
      VAR5                9 RD_1
      VAR6                B RD_1
    Input module : sam_asm2
      ext_m1           8500 CD_A     sam_asm1

Linkage editor end,With error
```

Chapter 3 Object Format

You can select one of five object formats, outlined below, to be output by the object converter TUCONV.

3.1 Intel Format

The Intel HEX format has 16-bit addressing, while the Intel extended HEX format has 20-bit addressing. The latter includes the extended address code added to the object format of the former.

```
Comment
:leng adr  type data ....data checksum
     :
:leng adr  type data ....data checksum
:leng sadr type checksum
```

leng : Number of data items in record (2-digit hexadecimal)

adr : "data" address (4-digit hexadecimal) following the "00" of "adr"

type : Record type

- 00 : Normal record
- 01 : End record
- 02 : Extended address record

data : One byte of data (2-digit hexadecimal)
The data in each record is contiguous.

checksum : Checksum (2-digit hexadecimal)
The hexadecimal values from "leng" to "checksum" are delimited into 2-digit values, each interpreted as one byte of data. This value is the complement of the lower 8 bits of the sum of these bytes.

sadr : Program execution starting address (4-digit hexadecimal)
This is the end record, and normally output as "0000". The type is "01".

Extended Address Records

Extended address records are of type "02" and address "0000". "data" shows the paragraph address (USBA). Data after encountering the extended address code is mapped to an address obtained by shifting the paragraph address 4 bits left and adding the address (adr) of the data record.

3.2 Motorola S Format

You can select 16-bit, 24-bit, and 32-bit addressing in the Motorola S Format. Each format includes a common header record plus data records with the respective address length and an end record for the respective data record.

S0:

```
type cnt adr data checksum
     :
type cnt adr data checksum
```

type : Record type
- S0 : Header record
- S1 : Data record (16-bit address)
- S2 : Data record (24-bit address)
- S3 : Data record (32-bit address)
- S7 : End record (32-bit address)
- S8 : End record (24-bit address)
- S9 : End record (16-bit address)

cnt : Number of bytes from "adr" to "checksum" (2-digit hexadecimal)

adr : Starting address of "data" (4-, 6-, or 8-digit hexadecimal)

data : 1 byte of data (2-digit hexadecimal)
Data in each record is contiguous.

checksum : Checksum (2-digit hexadecimal)
The hexadecimal values from "cnt" to "checksum" are delimited into 2-digit values, each interpreted as one byte of data. This value is the complement of the lower 8 bits of the sum of these bytes.

Header Record
The "data" item in the header record is a comment. The header record can be omitted.

Part 5 Error Messages

Chapter 1 Error Messages

1.1 Types of Error Message

There are three types of error message:

Warnings
Errors
Fatal errors

1.2 Error Message Format

Error messages take the following format:
<filename> <line number> : <tool>-<type>-<number> : <message>

```
test.asm 15: ASM870C-Error-200: Syntax error
```

<filename> This the name of the file in which the error occurred. No filename is displayed if the error was not related to a particular file. Normally, the filename is displayed for errors in C Compiler, macro processor, assembler processor, assembler, and linker command language files.

<line number> This is the number of the line in which the error occurred. Normally, this is displayed for errors for which the filename is displayed.

<tool> This is the name of the tool (assembler, etc.) in which the error occurred.

<type> This is the error type, described later.

Fatal
Error
Warning

<number> This is the error number, described later.

Fatal : 0 to 99
Error : 200 to 499
Warning : 500 to 999

<message> The message is a description of the error.

Chapter 2 Driver Error Messages

2.1 Fatal Errors of Drivers

<I/O Errors>

20: Can't open "<filename>"

The driver cannot open the specified file. Common problems are that the specified file does not exist, there is insufficient disk space to create a new file (when opened in write mode), or that the file is write-protected.

<Invocation Errors>

100: No source file found in invocation

No source file was specified in the startup command.

103: "<filename>" files are the same

The same filename is specified more than once.

106: Missing parameter "<option>"

A required parameter was not specified or was specified incorrectly.

109: Unrecognized option "<option>"

Wrong option specified.

111: Can't nest a command file

A command file was specified within a command file, but command files cannot be nested.

112: Not allowed character "<option>"

The option parameter is not a numerical value.

113: Invalid subargument "<option>"

Incorrectly specified option.

114: Invalid argument "<option>"

Incorrectly specified option parameter.

115: '-r' option requires '-o' option

You must specify a file output with the '-o' option when specifying the TULINK option '-r'.

116: Can't execute "<filename>"

The file specified in <filename> cannot be executed.

2.2 Warning Errors of Drivers

520: The suffix not fit for output-file "<filename>"

Error in output file extension (suffix).

521: Ignored option "<option>"

A reciprocal option was specified. The first specified option takes priority.

522: Unknown suffix, "<filename>" used as "<extension>" file

Error in specified input file extension.

Chapter 3 C Compiler Error Messages

C Compiler is invoked by CC driver. Please refer the error message of driver about CC driver.

3.1 Fatal Errors of C Compilers

<Runtime Errors>

100: Too many errors

The number of errors exceeded the compiler limit (30).

< File-related errors >

120: Cannot close file "<filename>"

The specified file cannot be closed.

121: Cannot open file '<filename>'

122: Cannot open temporary file

The specified file cannot be opened. Either the file does not exist or a file which cannot be specified was specified. Error 122 occurs when the file name is temporary file. Indicates that an intermediate file was not generated during compiling.

123: Cannot seek '(filename)'

An error occurred during file seek.

124: Problems with input file

An error occurred during file read. The file may be corrupted.

125: Problems with output file, probably out of disk space

An error occurred during file write. Available space in disk may not be enough.

< Compiler limits >

130: Compiler limit, out of space

Memory available for the compiler reached the limit. No more memory available for the compiler.

131: Compiler limit, too deep nesting of blocks

The number of nested blocks in the source program exceeded 16, its maximum value.

134: Compiler limit, too deep nesting of struct/union

The number of structure/union nesting levels exceeded 15, the compiler limit. Reduce the number of nesting levels by using typedef and defining a part of structure nesting as a different data type.

136: Compiler limit, too many internal variables in 'function'

The number variables or types exceeded the compiler limit.

137: Compiler limit, too many internal label

The number of internal variables exceeded the compiler limit because the function is too large. Break up the function before compiling.

138: Yacc stack overflow

Compiling the source program cannot continue because the work area for analyzing the source program is insufficient. Occurs when the source program is too complicated. Break up a large expression, especially an expression including a comma operator. Break the expression at a location where function calls are nested within an argument of a function call. Use a function for the nested part.

< Fatal errors due to memory insufficiency >

140: Out of memory

The area necessary for compiling cannot be obtained because system memory is not sufficient.

141: Out of near memory

Memory necessary for processing is not sufficient due to too many symbols and variables.

142: Too large function for optimization in '<function_name>'

Memory for optimization cannot be obtained. The function is too large to be optimized. Break it up and compiling may be possible.

< Limits exceeded >

150: Too large string

The length of a character string exceeded the size of the compiler buffer. The length of a character string after linkage is up to 512 characters.

151: Too large string for inline assemble

The character string specified in the inline assembly statement is too large.

< Other fatal errors >

160: Division by zero in '<function>'

161: Remainder by zero in '<function>'

Division by zero or remainder by zero is performed in the source program. Detected as a result of a constant operation. Division by zero or remainder by zero cannot be detected during program execution. If division by zero or remainder by zero is detected during preprocessing, 'Preprocessor' is indicated instead of 'function name'.

162: Memory control blocks destroyed

OS memory control blocks are destroyed. Re-start PC.

163: Unexpected EOF in comment

The file ended before the end of a comment. Comments cannot be written in more than one file, not can they be nested.

3.2 Errors of C Compilers

< Token errors >

200: Empty character constant

Null character constant "(two single quotations) was used as a character constant. A character constant must contain at least one character.

201: Illegal character '<Hexadecimal>'

202: Illegal digit '<character>' for base '<radix>'

A character whose character code is hexadecimal appeared in the source file. Such characters and Japanese characters cannot be used. The source file may be corrupted, so check the contents of the file.

203: Illegal escape sequence

An escape sequence is used in other than a character constant or a string literal.

204: Illegal hex constants

No hexadecimal character is written after 0x or \x indicating the start of a hexadecimal constant.

205: Newline in char constant

A newline character is input before the end of a character constant (before closing using '). Lines cannot be changed within a character constant.

206: Newline in string

A newline character is input within a string literal. When writing a string literal on two lines or more, write as follows: [By using a function to link logical lines] Add a backslash "\" or Yen mark "\" just before a newline character. [By using a function to link character strings] Write a character string enclosed between ". If there is only a space character (includes tab and newline) between character strings, the compiler links these character strings.

207: Unexpected exponent character '<character>'

A character other than a sign or numerical value is written after character "e" or "E" representing exponent character.

< Syntax errors >

210: Constant expected

A constant expression is required. For example, an expression other than a constant expression is written instead of an array size for array declaration.

```
int a = 5;
int b[a];
```

The above are wrong.

211: Illegal break

A break statement is written in a statement other than a do, for, while, or switch iteration statement.

212: Illegal continue

A continue statement is written in a statement other than a do, for, or while iteration statement.

213: initialization needs {} in '<identifier>'

An initializer for an array, structure, or union must be enclosed by braces {}.

```
int a[1] = 1;       /* wrong */
int a [1] = { 1 }; /* correct */
```

216: Syntax error at or near column '<column>'

A syntax error occurred. The number of columns is <column>.

< Declaration and definition errors >

221: Array of functions not allowed

An array of functions is not allowed. Nor can an array of *void* type be accepted. The pointer type to the function may have been incorrectly declared. The following shows an example of a pointer array to the function that returns the *int* type.

```
int (*f1[10])();
```

/* Correct: Pointer array to the function that returns the int type */

```
int *f1[10]();
```

/* Incorrect: Array of functions that return a pointer to int */

223: Cannot initialize extern '<identifier>' in block-scoped

A variable declared together with storage class specifier extern within a block is initialized.

224: Cannot use address of automatic variable as static initializer

An address of an object with automatic storage duration is used within an initializer to an object with static storage duration.

```
void func( void ) {
  int i;
  static int *p1 = &i; /* wrong */
         int *p2 = &i; /* correct*/
```

225: Duplicate signed/unsigned keywords

Both "signed" and "unsigned" are used within one declaration. Chose one.

226: Duplicate storage class specified

Two or more storage classes are specified within one declaration.

227: Expected formal-parameter list

An argument list instead of a parameter list is used for function definition.

```
/* wrong */          /* correct*/
void func( int )     void func( int arg )
{                    {
```

228: Function illegal in struct/union '<identifier>'

A function is declared as a structure/union member. Declaration of pointer type to a function may be wrong.

```
struct f {
    int (*f1)( void );
/* correct:  pointer to a function which returns int */
    int *f1( void );
}
/* wrong:  function which returns a pointer to int */
```

229: Illegal bit field type '<bit fileld>'

Illegal bit field type (such as pointer or floating point type) is specified. Bit field types are the following (to which signed or unsigned can also be added): char, short, int, long.

230: Illegal declaration '<storage class specifier>'

The specified storage class specifier cannot be used.

231: Illegal function return type, cannot return array type

Array type is specified as function return value type. Pointer type to an array can be specified as return value type.

232: Illegal function return type, cannot return function type

Function type is specified as function return value type. Pointer type to a function can be specified as return value type.

233: Illegal initialization

Initialization specification is illegal. Occurs when an initialization value is other than a constant expression or the comma operator is used in an initialization expression.

235: Illegal type combination '<type specifier>'

Some type specifiers cannot be used within the same declaration.

Given below is an example:

```
short long int i;
```

236: Illegal void type '<identifier>'

An attempt was made to declare a void type variable. Declaration of pointer (incomplete type) to a void type variable or void can only be used to declare a function without a return value or to indicate no arguments for function declaration.

237: Illegal zero sized member '<member>'

Array without size (array with no subscript, or subscript is 0) is declared as a structure/union member. An array without size can only be declared as the last member of a structure/union.

239: Negative subscript

A negative value is specified as the size in an array type declaration.

240: Non-address expression

An address is required for an initialization expression. An error will occur in the examples below:

```
int a;
int *b = a;
```

241: Non-constant initializer

The initializer is not a constant.

243: Null dimension

When defining a multi-dimensional array, other than at the first dimension, subscript values must be defined.

244: Prototype must have parameter types '<function>'

When declaring a function prototype, argument types must be specified using parameter type definition.

```
void func1( int arg1, int arg2 ); /* correct*/
void func2( int arg1, arg2 );     /* wrong */
void func3( arg1, arg2 );         /* wrong */
void func3();                     /* not prototype declaration */
```

The last example is an old-style function declaration. It is not wrong, but it is not a prototype declaration, either.

245: Too many initializers

Number of initializers larger than the number of objects to be initialized is specified.

246: Zero size bit field '<identifier>'

0 is specified as the width of a named bit field.

< Undefined errors >

250: Illegal struct/union name for member .'<member>'

The left operand of a component selection operator is not structure or union type. When the left operand is an undefined identifier, it is regarded as an int type variable. Thus, this error occurs.

251: Illegal struct/union pointer name for member ->'<member>'

The left operand of a component selection operator is not a pointer to structure or union type. When the left operand is an undefined identifier, it is regarded as an int-type variable. Thus, this error occurs.

252: Undefined struct/union '<identifier>' of '<tag>'

A structure/union with a <tag name> is not defined, therefore, <identifier> cannot be declared as a structure/union. This error also occurs when an attempt is made to initialize an unnamed union.

253: Undefined struct/union '<identifier>', left of '<operator>'

An undefined structure/union is used in the left operand expression of the component selection operator ("->" or ".").

254: Unknown size '<identifier>'

Array without size was declared as a temporary named variable (auto variable).

255: Unnamed first member of struct '<tag>'

Structure declaration starts with a member with an unnamed bit field.

256: '<identifier>' undefined

An attempt was made to use an undefined identifier. This error also occurs when a used label is not defined within a function.

< Double definition errors >

260: Duplicate case in switch '<value>'

The same value is used twice for case labels in one switch statement.

261: Duplicate default in switch

Two or more default labels are written in one switch statement.

262: Redeclaration of '<identifier>'

An attempt was made to define an already-defined <identifier>. Listed below are possible causes:

<identifier> is already declared as a tag name for structure/union or enumeration type.

The function of <identifier> was defined twice or more.

The object of <identifier> was defined twice or more.

<identifier> was declared twice or more in different types.

<identifier> was used twice or more in one structure/union or enumeration type.

The label of <identifier> was defined twice or more within one function.

263: Redeclaration of struct/union/enum/tag '<tag>'

An attempt was made to declare the <tag name> that had already been used in the struct, union, or enumeration declaration.

264: Redeclaration of struct/union member '<member>'

An attempt was made to declare members of the same name in the struct or union declaration.

< Operand errors >

270: Bad left/right operand '<operator>'

Operand type of the operator is illegal.

271: Illegal cast

An attempt was made to convert a type which cannot be converted. The type conversion in the example below is not allowed.

```
int a;
struct st{ int a1, b1;} st1;
st1 = (struct st)a;
```

272: Illegal indirection

Pointer operator "*" is used for a non-pointer value.

273: Illegal sizeof

The operand of sizeof operator must be either an object name or a type name.

274: Illegal struct/union type, use '->'

Component selection operator "." is used as a pointer to a member of a structure/union. When specifying the member of a structure/union pointed to by a pointer, use operator "->".

275: Illegal struct/union pointer type, use '.'

Component selection operator "." is used for a structure/union object. When specifying a member of a structure/union object, use operator ".".

276: Illegal subscript

Operator "[]" is used for an object of other than array or pointer type.

277: Unacceptable operand of '&'

The operand of address operator "l&l" is illegal. The following cannot be used as operands for the address operator.

bit field
register variable
value other than left-side value

278: '<member>' not member of struct/union

The <member name> is not a struct or union member.

< Expression errors >

280: Cannot cast void to non-void

Type void cannot be converted into another type. Type void indicates that the function does not return a value. A pointer to void type cannot be converted into a pointer to another type.

```
/* wrong: cast of void */
void func1( int arg );
int a = (int) func1( 1 );
/* correct: cast of pointer to void type */
void *func2( int arg );
int *a = (int *)func2( 1 );
```

281: Illegal actual parameter '<value>'th of '<function>'

At a function call, an argument has an error. <number> represents the argument number.

282: Illegal cast to array type

Object type is converted into array.

283: Illegal cast to function type

Object type is converted into function.

284: Illegal compare struct/union

Comparison between structures or unions is not allowed. Compare members of a structure or of a union.

285: Illegal function

A function is called using an identifier which is not declared as function type, or an expression which is not a pointer to a function.

286: Illegal index, non-integral

Value of an array subscript expression is not integer type.

288: Illegal operand '<operator>'

The operand is used incorrectly.

289: Illegal operator '%s' for struct/union

The structure or union cannot be operated on by the specified operator.

291: Incompatible types '<identifier>'

Operation on incompatible types is performed.

292: Invalid addition, pointer to pointer

Addition between pointer types is performed.

293: Invalid addition/subtranction, pointer to non-integral value

Addition or subtraction between pointer type and non-integer value is performed.

294: Invalid subtranction, pointer from non-pointer

Attempt is made to subtract a pointer type value from a non-pointer type value.

295: Lvalue required '<operator>'

The left operand of the operator must be a left-side value.

296: Lvalue specifies const object

Const type object value is used as a left-side value. Operation to change a const type object value is not allowed.

298: Non-integral in switch expression

The result of evaluating a switch expression is not an integer value.

300: Void type in expression

A void expression is used in a control expression of the if, while, for, or do statement. A void type function (function without a return value) cannot be used in a control expression. The expression type cannot be changed into void.

301: '<operator>' needs lvalue

The operand of the operator must be a left-side value.

< Preprocessor errors >

310: Cannot open #include file "<filename>"

The #include file <file name> cold not be found.

311: Illegal identifier '<identifier>' found in defined-operator

An unspecifiable <identifier> is used in the *defined* phrase of the #if or #elif statement.

313: Illegal macro name

An invalid identifier is specified as a macro name.

314: Illegal macro parameter '<parameter>'

An invalid character is specified as an argument in a function-type macro name definition.

316: Illegal #elif

The #elif is used incorrectly. The probable cause is that the corresponding #if, #ifdef, or #ifndef is missing.

317: Illegal #else

The #else is used incorrectly. The probable cause is that the corresponding #if, #ifdef, or #ifndef is missing.

318: Illegal #include filename

The file name specified in the #include command is incorrect.

319: Illegal #line filename

The file name specified in directive #line is illegal.

320: Illegal #line number

The line number specified in directive #line is illegal. Specify an integer from 1 to 65535.

322: Macro '<identifier>' redefined

Replacement contents differ in redefinition of the macro.

324: Too long token

The token name is excessively long. Specify it within 512 characters as an identifier or within 32 characters as a replacement character of the function format.

325: Too many #else

Usage of #else is incorrect. You've already used #else in the corresponding #if, #ifdef, or #ifndef.

326: Unexpected EOF in macro '<identifier>'

An EOF code is found in the macro call.

327: Unexpected EOF in #if/#ifdef/#ifndef

An EOF code is found in the #if, #ifdef, or #ifndef clause.

328: Unexpected #endif

The corresponding #ifdef is missing.

329: Unknown preprocessor command

The preprocessor command is invalid.

330: #elif following #else

The #elif command is written after the #else command.

331: #error '%s'

The #error command will be executed.

332: ')' not found in '<operator>'

Replacement contents differ in redefinition of the macro.

< Limits exceeded >

340: Too large bit field '<bit field>'

Number of bits larger than the standard number is specified at bit field declaration. For example, the specification below is illegal.

```
struct bitfield { int f1:40;  char f2:10; };
```

341: Too many characters in constant

A number of characters exceeding the size of the int type are specified in the character constant.

342: Constant too big

Constant value exceeds the range in which a value can be expressed by the specified type.

343: Out of range for enum constant

Enumeration constant exceeds the range in which a value can be expressed by int type.

< ANSI standards errors >

350: Bit field '<bit field>' must have type of int

Types which can be specified as bit field type in the ISO/ANSI C standards are as follows: int, signed int, and unsigned int. This error occurs only when compiling using option -Xa.

351: Function '<function>' storage class must be extern

A function is declared within a block, but extern is not declared. This error occurs when option -Xa is valid. Given below is an example of error generation:

```
main()
{
    static int func1();
     ...
}
```

352: Illegal lvalue

Since the operand is not a left-side value, it cannot be converted to the pointer type. This error occurs only when the -Xa option is used to compile the source files.

< Compiler limits exceeded >

360: Compiler limit, too deep nesting of #if/#ifdef/#ifndef

The nesting level of #if , #ifdef, and #ifndef exceeded 31, the compiler limit.

361: Compiler limit, too deep nesting of #include

The nesting level of # include exceeded 31, the compiler limit.

362: Compiler limit, too large array

The array size exceeded, the compiler limit.

363: Compiler limit, too many declaration

The number of qualifiers exceeds the compiler limit (12 levels).

364: Compiler limit, too many parameter

The number of arguments of a function exceeded 31, the compiler limit.

365: Compiler limit, too many -I options

The number of specifications of option -I exceeded 31, the compiler limit.

366: Compiler limit, too many -D options

The number of specifications of option -D exceeded 255, the compiler limit.

367: Compiler limit, too many -U options

The number of specifications of option -U exceeded 255, the compiler limit.

< Errors related to extension function >

370: Bad inline assemble construction

Syntax for _asm is illegal.

371: Duplicate function attribute

Two or more function qualifiers (eg, _cdec1) were specified within one declaration. For example, an error occurs to the following declaration:

```
typedef  int  __adecl  AINT( void );
AINT  __cdecl  func;
```

372: Illegal function call, function defined __interrupt or __regbank

_interrupt type functions cannot be called from a C program.

373: Illegal function return type

_interrupt type functions must be void or no specification (regarded as void).

374: Illegal parameter type list

_interrupt type functions cannot specify parameters.

375: Illegal pointer type, different function attribute

The pointer type is used erroneously because the function attribute is incorrect.

385: Illegal function type for inline/builtin-function '<function>'

The specified function cannot be made into the in line function.

< Others >

390: Division/remainder by zero

Divisor 0 or remainder 0 occurred during evaluation of a constant expression.

391: Static function '<function>' not found

Function declared as static was not defined. Function declared as static must be defined within one translation unit.

< Errors of preprocessor >

400: Illegal pointer size

The pointer size specification is illegal.

401: Illegal displacement size

The displacement size specification is illegal.

402: Illegal section size

The section size specification is illegal.

403: Illegal assignment of flag register '<register>'

A flag register cannot be assigned.

404: Duplicate memory size

Two or more memory sizes are included.

406: Cannot initialize io variable

An initial value cannot be specified for variable io.

407: Cannot declare io variable in block-scoped

Variable io cannot be specified within a block.

408: Not support keyword '__<keyword>'

The extended reserved word specified here is not supported.

3.3 Warnings of C Compiler

The number in parenthesis which follows to error messages shows a warning level.

(1) Level 1 Normal warning (Default -w1)
(2) Level 2 Middle warning (-w2)
(3) Level 3 Detail warning (-w3)

In warning level 3, when a program is not exactly written, the warning message is output. The description which may cause a mistake can be checked.

Note what the message meaning says, because the same message is shown with different levels according to circumstances.

< Warnings of option >

500: Duplicate option '<option>' (1)

The same option is specified twice or more.

501: Illegal option '<option>' (1)

Specification of the option is illegal. This option is ignored.

503: Option '<option>' requires an argument (1)

The parameter is not specified for the option which necessitates the parameter. This option is ignored.

< Warnings of discarding data >

510: Floating-point overflow (2)

A floating point operation exception such as overflow or underflow occurred during a constant operation on a floating point number at compile. This error also occurs when a constant is written exceeding the range which floating point type can handle. The compiler continues the operation, regarding the operation result as 0.0.

511: Out of range in hex escape sequence '<Hexadecimal_constant>' (1)

The number of digits in the hexadecimal escape sequence of a character constant or string literal exceeded 9 characters. A hexadecimal constant of 9 digits or more exceeds the range in which a value can be represented by long int; therefore, the conversion result may be incorrect.

513: Too many initialize for array '<identifier>' (1)

The number of initializers for the specified array is too large. The excess initializers are ignored.

514: Type conversion, possible loss of data (2)

Values of different base types were specified in one expression. The type of one of them was converted. During type conversion, data were discarded.

515: Too long identifier, truncated to '<identifier>' (1)

The identifier was too long; characters following the 32nd character were discarded. After characters were discarded, different identifiers may be regarded as the same.

516: Integral overflow (2)

The operation result exceeds the range which can be represented by integer.

517: Too big for character (1)

The internal code of the character constant exceeds the range which can be handled.

< Warnings of undefine >

521: Undefined return type in '<function>' (3)

A function which is not declared or defined is used. Compiling continues regarding the function as one which returns an int type value.

522: Unnamed struct/union as parameter (3)

This warning only occurs when a function without a prototype declaration is used. A tag name is not declared for a structure/union written in an argument for a function call. A structure/union without a tag name declared cannot be used for parameter declaration. A function with structure/union as a parameter cannot be defined. Thus, the function call may be illegal.

523: Undefined struct/union '<tag-name>' (3)

An undefined structure/union is used. They are compiled as a structure/union without members. (Undefined structure/union without identifiers is a warning of level 3.)

< warnings of unnecessary declaration/description >

530: Duplicate type qualifier (1)

The same qualifier (const or volatile) or type specifier (signed or unsigned) was used twice or more. For example, this warning is output in the following case:

```
const const int cint;
```

531: No identifier declared (2)

A null declaration is made, so it was ignored. Given below is an example of a null declaration:

```
int ;
```

532: Untagged enum/struct/union declared no symbols (2)

A null declaration using an untagged enumeration/structure/union is made. This declaration is ignored. For example, this warning is output in the following case:

```
struct {
   int m1, m2;
};
```

534: '<type>' may be used on integral types only (1)

Signed or unsigned is used for a type other than integer. The specification is ignored.

536: No use memory size (1)

Displacement and specifying I/O variables can not be executed.
(ex. auto variable, etc.)

537: Label '<label>' defined but not used in function (3)

A label which is not used is included.

538: Statement not reached (1)

A statement which is not executed is included.

< Descriptions which might cause errors >

540: Assignment in conditional expression (3)

The result of an assignment expression is used as a conditional expression. For example, this warning is output in the following case:

```
if ( result = func( 1 ) )...
```

Assignment operator "=" instead of equality operator "==" is often used by mistake. This warning is used to prevent such mistakes.

541: Case constant '<value>' too big for the type of switch expression (1)

Case label value exceeded the range in which values can be represented by the conditional expression type in the switch statement. The case label value is converted into conditional expression type. Note that values which are different before conversion may become the same as a result of the conversion.

542: Comma operator in array index expression (3)

Comma operator is used in an array subscript expression. For example, this warning is output in the following case:

```
array[dim1, dim2]
```

This warning is used to check whether a subscript expression is correct, because in some languages, multi-dimensional array subscripts are delimited using commas.

543: Constant in conditional expression (3)

A constant is specified in an if or while conditional expression. This warning is just for information.

544: Const object '<object>' should be initialized (3)

A value cannot be assigned to a const object unless the object is initialized using an initializer at definition. If the declaration of a const object is tentative, declaration "const it i;" causes this warning. If the initial value is specified by another translation unit, declare it explicitly as "extern const int i;". Then the warning will not be output.

545: Illegal assignment, const/volatile qualifier mismatch (1)

A pointer to an object which was declared as const or volatile is assigned to a pointer to an object which was not declared as const or volatile. As a result, the compiler may not be able to handle the object correctly.

```
const int a;
const int *cp = &a;
int *np = cp; /* warning */
*cp = 1;      /* error */
*np = 1;      /* not error */
```

In the last line in the above example, np is not a pointer to a const object, so the compiler cannot check assignment to const object a. To avoid this mistake, a warning is output at assignment to a pointer.

546: Illegal conversion, integral type mismatch (3)

During conversion of two integer values, data are lost. For example, this warning is output in the following case:

```
short v_short;
long  v_long;
v_short = v_long;
```

547: Illegal conversion, floating type mismatch (3)

During conversion of two floating point values, data are lost. For example, this warning is output in the following case:

```
float  v_float;
double v_double;
v_float = v_double;
```

548: Illegal pointer operation '<operator>', array's subscript mismatch (1)

A pointer to an array of different size is used in the expression. The pointer value was used unchanged.

549: Illegal pointer operation '<operator>', indirection level mismatch (1)

Contradictory indirect reference was made. The pointer value is used unchanged if both operands are arithmetic, pointer, or array type; or one of the operands is array type. If one operand is arithmetic type and the other is pointer or array type, the type of the arithmetic operand is converted into the type of the other. This warning is output in the following example; the pointer value is not changed but assigned as-is.

```
char **argv;
char *a;
a = argv;   /* warning */
```

550: Illegal pointer operation '<operator>', type mismatch (1)

There is a pointer to an object of a different type in the pointer expression used together with the above operator. The value was used unchanged. For example, this warning is output in the following case:

```
struct st1 *p1;
struct st2 *p2;
p2 = p1;   /* warning */
```

551: Logical operation '<operator>' on address of string constant (3)

A logical operation using a string literal address is performed. For example, this warning is output in the following example:

```
char *str = "Hello";
if ( str == "Hello" ) {...
```

In the above if statement, the contents of the string literal are not compared. This warning is used to prevent such mistakes. Use function strcmp to compare the contents of a string literal.

552: Meaningless statement (2)

Meaningless statement is written.

553: No return value in '<function>' (1)

A return statement is not written in a function which is declared for a return value (warning level 1). If the warning level is set to 3, this warning is output even if a return statement is specified that is used to return a value to a function regarded as int because of no return value type.

556: '<identifier>' used before set in '<function>' (3)

A variable which has had no value set is referenced.

557: Redeclaration of '<identifier>', array's subscript mismatch (1)

Different array elements are declared.

558: Multiple comparison operator in expression (3)

Comparison operations are written twice or more.

559: Out of range for array (3)

Accessing is executed out of the range for the specified array.

< Warnings to specifications >

560: Bad storage class '<identifier>' ***(1)***

A storage class identifier which cannot be used is specified. The compiler continues processing regarding one of the following default storage classes as specified.

function	extern
parameter, local variable	auto
global variable	no storage class specification

561: Cannot return value for void function ***(1)***

A return statement is written within a function declared as void, which does not return a value.

563: Storage-class specifier after type ***(3)***

Storage class specifier, auto, extern, register, or static, is declared after the type specifier. For example, this warning is output in the following declaration:

```
short int auto i;
```

In the new style, the storage class specifier is specified at the beginning. However, the compiler regards the storage specifier as at the beginning of the declaration even if it was not.

564: Illegal zero sized member '<identifier>' ***(3)***

The specified structure/union member does not have a subscript or includes an array with subscript 0. (level 1)

The specified structure/union member does not have a subscript or includes only an array with subscript 0. (level 3)

565: Redeclaration of '<function>', class mismatch ***(3)***

The function is defined in different storage class again.

When an extension specification is used, not errors but level 3 warning is output.

In ANSI specification, an error is output.

< Warnings for prototype >

571: Illegal declaration with formal argument list ***(1)***

Parameters are not declared for definition of a function which was declared as using parameters. In subsequent function calls, the function is regarded as not using arguments.

572: Illegal assignment of function's pointer, different parameter lists (1)

A value is assigned to a pointer to a function. The parameter lists of the function pointed to by this pointer and by another pointer are different. The compiler continues without changing the value.

573: Illegal declaration without formal argument list (1)

Parameters are declared in the definition of a function which was declared as not using arguments (void). After the definition, the function is regarded as having arguments defined.

574: Illegal function call '<function>', declared with void (1)

A pointer type specified as an argument differs from the pointer type given to a parameter at function declaration or definition. The argument value is passed unchanged.

577: Illegal function call '<function>', too few actual parameters (1)

The number of arguments specified for a function call is smaller than the number of parameters declared at function declaration or definition. Only the specified arguments are passed to the function, therefore, the function may not operate correctly.

578: Illegal function call '<function>', too many actual parameters(1)

The number of arguments specified for a function call is larger than the number of parameters declared at function efinition. The excess arguments are passed to the function according to the rules for function calls.

579: Illegal function call '<function>', type conversion (1)

The base type of arguments differ from that of parameters. The base type of arguments is converted into that of parameters; however, data may be lost due to the conversion.

580: Illegal prototype in '<number>'th parameter (1)

The same function is declared twice or more but the parameter list types do not match. <number> indicates the location (from the start parameter) of the parameter whose type is different.

581: Different declaration parameter list from definition (1)

The type of the argument list at function declaration differs from the type of the parameter list at function definition. The parameter list at function definition is used instead of the argument list at function declaration.

582: No function prototype '<function>' (2)

A function whose prototype is not declared is called. Compiling is performed to call a function without prototype declaration according to the rules for defaults.

583: Parameter type mismatch, '<value>'th parameter of '<function>' (1)

An argument of a different type was passed instead of the parameter specified at function definition

584: Parameter number mismatch in prototype (1)

The same function is declared twice or more. The number of parameters is different for each declaration.

585: Uses old-style declarator '<function>' (3)

Function declaration and definition are old-style. This warning is output to check old-style declaration to avoid passing the wrong number of arguments or type because old-style declarations have no parameter type data.

```
int strlen( const char *string );
/* new-style function declaration */
int strlen( const char *string )
{
    ...
}
/* old-style function declaration */
int strlen();
/* old-style function declaration */
int strlen( string )
    char *string;
{
    ...
}
```

586: No use in formal-parameter list '<identifier>' (1)

When function definition is old-style, a type which is not used is defined.

587: Illegal function's pointer type, different return type (1)

The function pointer is used erroneously because the type of return value is incorrect.

< Warnings of preprocessor >

591: Illegal/missing macro name (1)

An invalid macro name is specified or the macro name is not specified.

592: Illegal macro call '<macro name>', mismatched number of paramters (1)

The number of arguments in a function format macro call does not match the number of arguments in the macro definition.

596: Illegal ##, beginning of a macro definition (1)

The macro definition replacement list starts with a ## operator.

597: Illegal ##, ending of a macro definition (1)

The macro definition replacement list ends with a ## operator.

598: Macro formal parameter expected after # (1)

The operand after a # operator in a macro definition must be a parameter name.

599: Unexpected character after directive, ignored (1)

An invalid character string is specified after a macro processor directive. The character string is ignored.

< Warnings of progra >

610: #pragma keyword expected, '<token>' found (1)

<token> after #pragma was not identified as a command. #pragma directive is ignored.

611: #pragma [on | off] expected (1)

#pragma directive needs an on or off parameter, but the parameter was not specified or the specified parameter was not identified. This directive is ignored.

612: #pragma [1 | 2 | 4] expected (1)

#pragma directive needs a 1, 2, or 4 parameter, but the specified parameter was not identified.

615: Unexpect #pragma token '<token>'

An unnecessary token was found in the argument list of #pragma directive. The remaining part of this directive is ignored.

616: Cannot use #pragma disinterrupt for inline/builtin-function (1)

A disinterrupt function definition is specified for the inline function. Since this function expands code directly, disinterrupt cannot be specified for it. This #pragma command will be ignored.

618: Unknown #pragma (1)

#pragma directive not supported by the compiler is used. This directive is ignored.

619: Cannot use #pragma in initializing (1)

#pragma directive is used in the middle of initializing.

< Others >

620: Cannot use function attribute '<identifier>' (1)

An identifier is declared as a non-function or a non-pointer to a function, but a function qualifier was found in the declaration. The function qualifier is ignored.

621: Illegal escape sequence '<character>' (1)

<character> after a backslash "\" or "´" in a character constant or string literal cannot be used as an escape sequence. The character code is used unchanged as the value.

622: Sizeof returns 0 (1)

The size of the operand of the sizeof operator is 0.

623: Type definition in formal parameter list '<tag>' (1)

624: Type definition in formal parameter list (no tag) (1)

Structure/union/enumeration type is declared in a formal parameter list. This declaration is regarded as an external declaration. If untagged structure/union/enumeration type was declared, the tag name is indicated as "no tag".

626: '*/' found outside of comment (1)

"*/" is written outside a comment. The compiler assumes a space between "*" and "/" and continues processing. For example, the warning is output in the following case:

```
int  */* comment */ptr;
```

After processing the above, the following is regarded as:

```
int  *ptr;
```

631: '/<character>' found inside of comment (3)

"/*" and "//" are found in a comment.

This description is ignored.

632: Static function '<function>' not found (1)

Static function which has no entities is found.

633: Illegal assignment of register (1)

The value of pseudo-variables may be invalid.

634: Too large function `<function_name>', optimize not performed (2)

Several optimization cannot be done because the function is too large.

635: Too much register pseudo-variable use, value may be invalid (1)

The value may be invalid because of the pseudo-variables are used too much.

<Warnings of processor dependent extension function>

650: Illegal pointer size (1)

A pointer, which cannot be used, is specified.

651: Illegal displacement size (1)

A displacement, which cannot be used, is specified.

652: Illegal section size (1)

A section, which cannot be used, is specified.

653: Illegal pointer operation '<operator>', pointer size mismatch (1)

In a pointer expression used with this operator, a pointer, which is used for a different object type, is found.

Chapter 4 Assembler Error Messages

4.1 Assembler Fatal Errors

<I/O Errors>

20: Can't open "<filename>"

Assembler preprocessor cannot open the specified file. Common problems are that the specified file does not exist, there is insufficient disk space to create a new file, or that the file is write-protected (when opened in write mode). Note that, when a work file is specified, the file is created in the directory (drive) indicated by the TMP environment variable.

21: Can't close "<filename>"

Cannot close the specified file.

22: Can't read "<filename>"

Cannot read the specified file.

23: Can't write "<filename>"

Cannot write to the specified file. The main cause of this error is that there is insufficient space for the file. Note that, when the file is a working file, it is created in the directory (drive) indicated by the TMP environment variable.

24: Can't seek "<filename>"

Cannot perform a seek on the specified file.

<Invocation Errors>

100: No source file found in invocation

No source file was specified in the startup command.

101: Illegal file specification

The file specification is illegal..

102: File must be a disk

You cannot specify a file other than a disk file. CON, PRN, AUX, COM1 and COM2 have specific meanings under OS and cannot be specified.

103: "<filename>" files are the same

The same file name was specified more than once.

104: Duplicated source file name

The source file name was duplicated.

105: Bad parameter syntax

A parameter of an option contravenes the entry rules.

106: Missing parameter "<option>"

A parameter which should be specified for an option is missing.

107: Illegal sub option in '-l'

The '-l' suboption specification is invalid.

109: Unrecognized option "<option>"

An invalid option was specified.

110: Numeric constant out of range

The numeric value is out of range.

111: Can't nest a command file

The command file is nested.

<Execution Errors>

150: Internal(system) error

An internal error other than those listed below occurred. This should not normally occur.

152: Illegal source format

The source file format is not suitable for environment.

154: Internal object file error

The object file is abnormal.

155: Too many expressions

Too many expressions or expressions are too complex. Combine the expressions into simpler expressions.

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 source file being assembled into smaller files or increase the amount of memory.

161: Out of memory

The working memory area was insufficient. Divide the source file being assembled into smaller files or increase the amount of memory.

163: Too many "file" instructions

"file" instructions of debug information are too many.
Decrease the include files of one C source file.

4.2 Assembler Errors

200: Syntax error

An error other than those listed below occurred.

201: Attempt to divide by zero

A divide by zero occurred.

202: Illegal numeric constant

A numeric constant was invalid. Common causes are specifying characters which cannot be used in numeric expressions.

203: Multi-defined symbol "<symbol>"

The symbol is already defined. Re-definition or multiple definition is not allowed. This definition is ignored. When accompanied by an instruction to output the object, '00' is output.

204: Invalid relocatable expression

An invalid relocatable expression. The majority of relocatable expression operations are permitted, but in operations, such as specifying the number of shift bits in an operand shift operation, a relocatable expression cannot be used.

205: Unbalanced parentheses

A parentheses structure is invalid.

206: Invalid expression

Invalid expression.

208: Illegal label or variable

Invalid label or variable.

209: Illegal character string

Invalid character string.

210: Not allowed public attribute

Directive names such as section names cannot be declared as PUBLIC. The instruction is ignored.

212: Illegal SECTION directive

An invalid SECTION directive. The section definition is ignored and the previous section continued.

215: No section definition

Instructions are entered without a section definition.

216: Invalid section attribute

The section attributes of a directive such as SECTION and EXTERN is invalid. The instruction is ignored.

217: Absolute section error

Absolute section addresses overlap.

218: Illegal control

Invalid control statement. The control statement is ignored.

220: Reference to multi-defined symbol

A multiply defined symbol was referenced.

221: Undefined symbol

An undefined symbol was referenced.

222: Absolute expression expected

A relocatable value cannot be used. the instruction is ignored.

223: Not allowed forward reference

Forward references are not allowed. the instruction is ignored.

225: Not allowed section reference

Section names can not be referenced.

226: Illegal symbol reference

An invalid reference. For example referencing a symbol from a different section.

227: Out of range for relative reference

A relative branch instruction exceeds the branch range.

228: Overflow in location counter

A location counter exceeds the section size specification range. The location counter counts up unchanged and processing is continued.

229: Location counter can't point lower address

The specified address value is less than the current location counter. The instruction is ignored.

230: Operand type mismatch

The operand, or mnemonic and operand, types do not match.

231: Too few or many operands

Too few or too many operands were specified.

232: Section "<section_name>" does not exist

The section specified as a sizeof or startof operator parameter does not exist.

234: The nesting level isexceeded

Files are nested exceeding the maximum nesting level.

300: Illegal operand value for CALLV or CALLP

An operand value of CALLV or CALLP is out of range.

4.3 Assembler Warning Errors

500: Illegal string constant

Invalid string constant. The string constant exceeds four characters. The fifth and subsequent characters are discarded.

501: Operand value is out of range

The operand value is out of range. The required number of bits, from the least significant, are extracted. Consequently, negative values may sometimes be converted to positive values.

502: Too long title

The title exceeds 60 characters. Only the first 60 characters are significant.

503: Some optimizations lost

Non-optimized labels remain.

504: Invalid instruction in this section

Invalid position for a machine instruction. The machine instruction itself will be correctly assembled. For example, if a machine instruction is entered in the data section, object code will be produced in the data section and the data section location counter updated.

505: Invalid directive in this section

Invalid position for a directive. The directive itself will be correctly assembled. For example, if a dt or dft directive is entered in the code section, data will be produced in the code section and the code section location counter updated. The same applies to ds directive outside the data section.

506: No END directive

No END directive at the end of the source file. It is treated as if the directive was present.

507: Text found after END statement

Source statements continue after the END directive. Source statements after the END directive are not assembled. Only the source lines are output in the assemble listing.

508: Invalid value in ALIGN directive

A numeric value is out of range in an ALIGN directive.

509: Duplicated MODULE directive

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

510: Ignored this TITLE

As NOLIST or similar was specified, this TITLE was ignored.

511: Ignored this EJECT

As NOLIST or similar was specified, this EJECT was ignored.

512: SAVE stack overflow

SAVE instruction nesting has exceeded eight levels. The SAVE instruction was ignored.

514: Source file empty

No source statements. A list file containing only the header is output.

515: Illegal parameter

Invalid parameter specification or multi defined parameter. The parameter is ignored.

516: Illegal escape sequence

Invalid escape sequence. The escape sequence is ignored. When it is a numeric value the higher order is discarded.

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 continued.

518: The floating-point type is not correct.

The floating constant type is not correct.

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

The symbol table cannot be sorted due to lack of memory. Processing continues without sorting the symbol table.

Chapter 5 TULINK Error Messages

5.1 TULINK Fatal Errors

<I/O Errors>

20: Can't open "<filename>"

Cannot open the specified file. Common problems are that the specified file does not exist, there is insufficient disk space to create a new file, or that the file is write-protected (when opened in write mode). Note that, when a work file is specified, the file is created in the directory (drive) indicated by the TMP environment variable.

21: Can't close "<filename>"

Cannot close the specified file.

22: Can't read "<filename>"

Cannot read the specified file.

23: Can't write "<filename>"

Cannot write to the specified file. The main cause of this error is that there is insufficient space for the file. Note that, when the file is a working file, it is created in the directory (drive) indicated by the TMP environment variable.

24: Can't seek "<filename>"

Cannot perform a seek on the specified file.

<Invocation Errors>

100: No source file found in invocation

No source file was specified in the startup command.

101: Illegal file specification

The file specification is illegal.

103: "<filename>" files are the same

The same file name was specified more than once.

104: Bad parameter symtax

A parameter of an option contravenes the entry rules.

105: Missing parameter "<option>"

A parameter which should be specified for an option is missing.

106: Illegal sub option in '-l'

The '-l' suboption specification is invalid.

108: Illegal character "<character>"

The character is not recognized as an option.

110: Unrecognized option "<option>"

An invalid option was specified.

111: Illegal numeric constant

The numeric value is out of range.

112: '-r' option requires '-o' option

When the '-r' option is specified the output file must also be specified via the '-o' option.

113: Both '-r' and '-ng' are set.

When the '-r' option is specified the '-ng' option cannot be specified.

<Execution Errors>

120: Bad object format in "<filename>" (<address>)

The input file is not of the correct object file format.

121: Illegal processor name in "<filename>"

The object format processor name is not the specified name. This error mainly occurs when a file of other than TULINK object file format is specified.

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

An invalid symbol class is present.

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

An invalid relocation type is present.

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

The symbol is already bound to an output section.

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

The location attributes were different for the sections to be combined.

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

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

132: Malformed "<section_name>" of "<filename>"

The section format is abnormal. An attempt has been made to link the wrong file or the file has become corrupted.

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.

140: Can't link "<section_name>"

Cannot link this section.

141: Run is too large and complex

Memory location failed because the location specification was too complex.

142: Yacc stack overflow

yacc stack overflowed.

<Command Language Errors>

150: Syntax error

The syntax of the directive on the specified line number is incorrect.

151: Illegal section name or memory name

Invalid memory name or section name. A number or a reserved word was specified.

152: Illegal address as origin or length

Invalid memory start address or length.

153: Memory specification ignord

The MEMORY directive specification contains an error. The instruction was ignored.

154: Multiple reference of a input section

A single section is referenced multiple times in a SECTION directive.

155: Illegal assignment

The assignment statement contains an error or is too complex.

156: Semicolon required after expression

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

157: Bad fill value

No padding value was specified in the -F option, or a value of two or more bytes was specified.

158: Multiple defined memory "<memory_name>"

The same user defined memory is defined multiple times in the MEMORY directive.

159: "<section_name>" can't be given an owner

The SECTION directive location address specification and output memory specification are inconsistent.

160: Output specification ignored

Invalid output specification in the SECTION directive. The main cause of this error is the multiple definition or incorrect sequence in the output specification 'org', 'align', 'len', 'addr' and 'type'.

161: OVERLAY section must BINDed

An address specification is required for an OVERLAY section.

163: Statement ignored

The statement contains an error and was ignored.

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

Invalid expression. The main cause of this error is when undefined or unresolved output sections are specified for the privileged operators org, addr and sizeof.

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

Memory areas specified in the MEMORY directive overlap.

169: Illegal operator in expression

The expression contains an invalid operator.

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

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

172: Can't nest a command file

The command file is nested. Or a file other than a relocatable object file or library file was specified in the command file specified as a parameter.

173: Illegal expression

The expression contains an error. This error occurs in cases such as when the expression contains the addition of two or more relative values.

5.2 TULINK 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 directive (at <address>). The main cause of this error is when the memory area defined by the MEMORY directive was too small or when input section type memory was not defined even though there was a MEMORY directive. This error usually occurs together with Error 211.

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

Sections of different types (CODE, DATA) cannot be combined.

203: Absolute sections can't in SECTIONS

Absolute sections cannot be specified in the SECTION directive.

205: Can't allocate '<section_name>'

The output section could not be allocated.

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

The memory allocated to the output section is full.

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

The memory allocated to the input section is full. This error usually occurs together with Error 201.

212: Symbol "<symbol>" attribute mismatch

An externally defined symbol or externally referenced symbol of the same name has a different attribute.

213: Can't find section "<section_name>"

Cannot find the input section of the specified name.

214: "<section_name>" not yet allocated

The section is not allocated.

216: Misuse of DOT

Invalid use of '.'.

217: Value of "<symbol>" in "<filename>" not fit in the object code

The symbol value after relocation is outside the object code size.

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.

220: Section "<section_name>" is too big

The input section size is too large.

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 attributes.

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 size and public declaration size are different.

233: Section "<section_name>" at "<address>" attempt to divide by zero

A divide by zero occurred at location "<address>" in "<section>".

234: Can't allocate medium sections(Small Data Area)

The link command file does not have DATA.M or ROMDTA.M attribute memory

defined in it. For sections that have DATA.M or ROMDTA.M attribute to be located in memory, the link command file must have memory that bears DATA.M or ROMDTA.M attribute defined in it.

235: Medium section("<section_name>") must be allocated fromt "<address1>" to "<address2>"

Sections that have DATA.M or ROMDTA.M attribute must be located in areas represented by <address 1> to <address 2>.

236: Medium memory (Small Data Area) too big, maximum 64k bytes

Sections to be located in memory that bears DATA.M or ROMDTA.M attribute must be located within 64 Kbytes beginning with the start address whichever lower (smaller).

5.3 TULINK Warning Errors

500: Absolute symbol "<symbol>" being redefined

An absolute symbol 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 size.

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 directive was ignored because the '-r' option (create a relocatable output file) was specified.

511: Unresolved external symbol "<symbol>"

An unresolved external symbol was found.

512: Duplicate execute address definition

An execution address has been specified more than once. The first specification will be used.

513: Section "<section_name>" size lager than definition

The output section size exceeds the size specified in the command language.

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. This message is output together with Error 207. The value of "<symbol>" specified in "<filename>" is used.

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

The symbol is multiply defined. This message is output together with Warning 500 and Warning 501. The value from the last specification of "<symbol>" in the command language file is used.

517: Useless symbol definition with '-r' option

An assignment statement is defined in incremental linking.

518: Illigal parameter

An illegal option is specified.

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

The pre defined memory is multiply defined or overlapped.

522: No debug information exist in <input_filename>

There is no debug information data in the specified file.

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

The start address "addr" 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.

525: Size attributes between input section and 'addr' are mismatch

The size attribute of the input section does not match that of the area whose start address is indicated by addr.

Chapter 6 TUMPP Error Messages

6.1 TUMPP 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

Symbol is not specified after -D or -S option.

009: Command line characters exceed maximum limit

The number of characters of command line exceeds maximum limit.

010: Same filename <filename> specified

The same filename is specified.

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

Filename has not been specified with given option.

012: Source file not specified

Source filename has not been specified.

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

The number of arguments of -e, -f, -mf, -GN option exceeds maximum limit.

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

The argument length of option argument exceeds maximum limit.

017: Source filename exceeds maximum limit

The input filename length exceeds maximum limit.

018: Output filename exceeds maximum limit

The output filename length exceeds maximum limit.

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

Argument is needed for <option name>.

<General>

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

The number of characters of a line exceeds maximum limit.

041: Number of source lines exceeds limit

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

042: Out of memory

Memory area cannot be allocated.

<Preprocessor>

080: '#endif' expected

Before terminating an #if, #ifdef or #ifndef directive with a #endif directive, end of file was found.

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

An identifier must be 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 filename which is specified as #include cannot be opened.

086: Unexpected '<reserved word>'

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

087: '#line' filename exceeds maximum limit

The length of filename of #line exceeds maximum limit.

088: '#error' message text exceeds maximum limit

The length of message of #error exceeds maximum limit.

<Macroprocessor>

None.

<Lexical>

160: Unexpected EOF in block comment

EOF occurs in block comment.

6.2 TUMPP Errors

<Command Line>

None.

<General>

240: ***Syntax error***

An invalid syntax has been specified.

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***

Character constant has more than 4 characters.

245: ***No character in a character constant***

No character is specified as character constant.

246: ***Illegal expression***

Invalid expression is specified.

<Preprocessor>

280: ***'#include' filename exceeds maximum limit***

The length 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***

A macro definition cannot begin with a token pasting operator (##), since a token pasting operator requires two tokens, one before it and one after it.

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

A macro definition cannot end with a token pasting operator (##), since a token pasting operator requires two tokens, one before it and one after it.

284: ***Formal parameter missing after '#'***

The token following a stringizing operator (#) must be a parameter.

285: ***#error : <string>***

TUMPP has encountered #error directive and has displayed the given message <string>.

286: ***Invalid line number for '#line'***

The #line directive encounters a invalid line number.

287: '#line' expects string as filename

The #line directive is not specified with required filename specification.

288: '#undef' expect an identifier

Macro name is not specified in the #undef directive.

289: Unexpected end of line

Unexpected end of line encounters. Correspondence of parenthesis may be not correct.

290: '#include' expect a filename

An #include directive is not specified with required filename specification.

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 does not accord with parameters.

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

Reserved word cannot be redefined.

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

Predefined symbol cannot be redefined.

295: Redefinition of parameter name <parameter>

The parameter names cannot be 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>'

Predefined symbol 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 has been already defined.

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

Reserved word has been specified for definition.

323: Redefinition of parameter name <parameter>

There are duplicate arguments.

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

There are duplicate local symbols or labels.

325: Macro function nest overflow

Number of nesting levels exceeds the 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>

Number of arguments specified in the macro call is more than or less than the number of arguments specified in the definition.

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

Macro call has not been specified in the new line.

331: Repeat value <expression> out of range

Value of the expression specified for ?repeat function is out of the range.

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

Macro definition has not been specified in the new line.

333: Valid trigger character expected

Trigger character specified is not a valid trigger character.

334: '?elseif' after '?else'

?elseif keyword used after ?else .

335: '?else' after '?else'

?else statement used multiple times for a ?if .

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

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

337: '?endif' without '?if'

?endif specified without specifying ?if .

338: '?restrict_macro' has already been specified

The ?restrict_macro has been specified more than once.

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

?endres can be specified only after a ?restict_macro has been specified.

340: '?endres' has not been specified

?endres has not been specified after ?restict_macro.

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

Invalid function is used.

342: Unexpected '?delayslot' is specified

?delayslot cannot be used at specified place.

343: Number of macro expansion exceeds maximum limit

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.

<Lexical>

None.

6.3 TUMPP Warning Errors

<Command Line>

500: Extra source file ignored

Only one source file can be specified.

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

A command line option has been specified twice. As specifying duplicate command line option is not valid, only the first option specified is considered valid and the second option is ignored.

502: Option '<option name>' conflicts with '<option name>', first option is valid

Two conflicting command line options have been specified. As specifying conflicting command line option is not valid, only the first option specified is considered valid and the second option is ignored.

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

Option is ignored because list file option is not set.

<General>

540: Expression greater than 32bit length, excess ignored

The value of expression is overflow.

541: Decimal number has octal prefix

Octal number includes 8 or 9. Or decimal number starts with 0.

<Preprocessor>

580: Unexpected characters following directive '<directive>'

Extra characters are found after processing a preprocessor <directive>.

581: Macro <symbol> redefined

The given macro identifier is defined more than once.

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

The number of characters in the replacement string of the preprocessor directive #define exceeds the maximum limit.

<Macroprocessor>

None.

<Lexical>

660: Identifier too long, excess truncated

When the identifier consist of number of characters greater than the allowable limit.

Chapter 7 TULIB Error Messages

7.1 TULIB Fatal Errors

20: Can't open "<filename>"

Cannot open the specified file. Common problems are that the specified file does not exist, there is insufficient disk space to create a new file), or that the file is write-protected (when opened in write mode. Note that, when a work file is specified, the file is created in the directory (drive) indicated by the TMP environment variable.

22: Can't read "<filename>"

Cannot read the specified file.

23: Can't write "<filename>"

Cannot write to the specified file. The main cause of this error is that there is insufficient space in the file. Note that, when the file is a work file, it is created in the directory (drive) indicated by the TMP environment variable.

24: Can't seek "<filename>"

Cannot perform a seek on the specified file.

25: Can't creat "<filename>"

Cannot create the specified file.

26: Can't creat temp file

Cannot create a work file.

27: Can't open command file "<filename>"

Cannot open the specified command file. The main cause of this error is that the file does not exist.

28: "<filename>" is not reading permited

The specified file does not have read permission.

29: "<filename>" is not writing permited

The specified file does not have write permission.

30: Cannot return current directory

The directory being accessed does not exist. It is possible that the current directory was deleted during execution.

<Invocation Errors>

110: Unrecognized option "<option>"

An invalid option was 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 and l must be specified.

117: Only one of [drtl] allowed

Incompatible options were specified. No more than one of the options d, t, r and l can be specified at one time.

118: Cannot use option in commandfile

An option which cannot be used in a command file was specified.

119: Target processer is different

The object files or library files target processor families do not match. For example, this error occurs when a TLCS-870 object is added to a TLCS-900 library file.

125: Option '-r' is libraryfile or objectfile

126: "<filename>" not in library format

The specified file is not in library format. The main cause of this error is specifying a non-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 could not be reserved. Divide the library files into smaller files and retry or increase the amount of memory.

136: Time of "<filename>" is broken

The time read from the specified file is incorrect. The file contents are invalid, so recompile or reassemble is required.

174: Cannot appoint module of "<module_name>"

An object file cannot specify a module. Object files must specify modules by file name rather than by module name.

7.2 TULIB Errors

240: Cannot open "<filename>"

Cannot open the specified file.

241: "<filename>" not found

Cannot find the specified file.

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.

246: "<filename>" is not reading permited

The specified file does not have read permission.

247: "<filename>" is not writing permited

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.

249: Cannot appoint Library file

Cannot recognize as a library file. The main cause of this error is specifying a non-library file as a library file.

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

The specified file is not an object file. The main cause of this error is specifying a non-object file as an object file.

Chapter 8 TUCONV Error Messages

8.1 TUCONV Fatal Errors

<I/O Errors>

20: Can't open "<filename>"

Cannot open the specified file. Common problems are that the specified file does not exist, there is insufficient disk space to create a new file, or that the file is write-protected (when opened in write mode). Note that, when a work file is specified, the file is created in the directory (drive) indicated by the TMP environment variable.

21: Can't close "<filename>"

Cannot close the specified file.

22: Can't read "<filename>"

Cannot read the specified file.

23: Can't write "<filename>"

Cannot write to the specified file. The main cause of this error is that there is insufficient space for the file. Note that, when the file is a workig file, it is created in the directory (drive) indicated by the TMP environment variable.

24: Can't seek "<filename>"

Cannot perform a seek on the specified file.

<Invocation Errors>

100: No source file found in invocation

No source file was specified in the startup command.

101: Illegal file specification

The file specification contravenes the rules.

102: File must be a disk

You cannot specify a file other than a disk file. CON, PRN, AUX, COM1, and COM2 have specific meanings under OS and cannot be specified.

103: "filename" files are the same

The same filename was specified more than once.

104: Duplicated source file name

The same source file name has been defined.

105: Bad parameter syntax

A parameter of an option is used incorrectly.

106: Missing parameter "<option>"

A parameter required for an option was missing.

107: Illegal sub option

The suboption specification is invalid.

108: Not parameter allowed

A parameter was specified for an option which does not take parameters.

109: Unrecognized option "<option>"

An invalid option was specified.

110: Can't nest a command file

The command file is nested.

111: Not supported option "<option>"

Unsupported option(-r, -n, -p) is specified.

112: Illegal output file "<filename>"

Wrong output file name specified. 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

Wrong numeric value specified.

114: Ambiguous point in block definition

The block specified with -ra option has ambiguous points. The block specified range may include multiple sections. In this case, the specified with -rb option is valid.

115: Can't find section "<section_name>"

No section specified with -rb option.

116: Can't find section "<section_name>"

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

150: Not an absolute object format

The input is not in absolute object format.

151: Bad object format

The object format of the input file is invalid.

152: Too large address

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

153: Load address overflow

An address exceeded the upper address limit for the specified format.

154: Out of memory

Working memory is insufficient. Increase the amount of memory.

155: Address overlap in "<filename>"

Address specification overlaps in the output range.

8.2 TUCONV 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

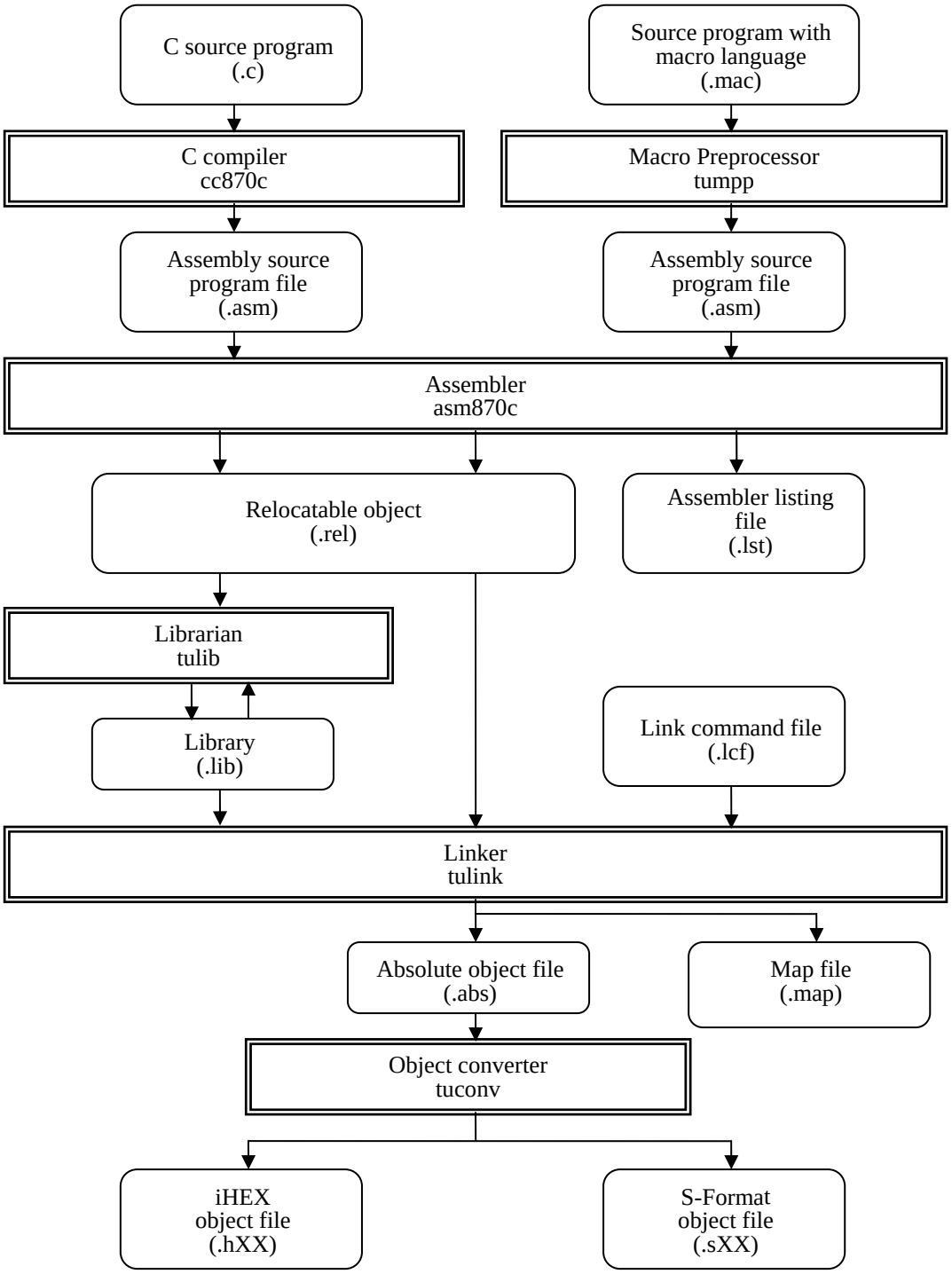
An invalid parameter.

516: Ignored option "<option>"

Ignored option specification

Appendix

Appendix A System Flow



Appendix B History

The history of this manual is the following.

Issue	Date	Update
1st	Nov 1, 2002	1'st Edition
2nd	Nov 24, 2003	Deleted Built-in-function Deleted MAC Driver Fixed miss-spelling
3rd	Nov 1, 2004	Deleted C-Like Compiler Fixed miss-spelling
4th	Jun 5, 2007	Deleted Tumpl Deleted Tuapp Deleted Tufal Added Tumpp Added TLCS-870/C1 specifications Fixed miss-spelling

TLCS-870 Family Language Tools Operation Guide [4th Edition]

The Date of Issue: 5 Jun, 2007