User's Manual

DSPASM

FAA/GREEN_DSP Structured Assembler

User's Manual

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.05 Dec 2022

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

© 2022 Renesas Electronics Corporation. All rights reserved.

Table of Contents

1. Overview 5

2. About DSPASM 6
2.1 Operating Environment 7
2.2 Input to the DSPASM 7
2.3 Output from the DSPASM 9
2.4 Command-Line Options for the DSPASM 12

3. Overview of Preprocessing 23
3.1 Distinguishing Identifiers for Preprocessing 23
3.2 Macro Replacement 23
3.3 Conditional Inclusion 26
3.4 File Inclusion 31
3.5 Predefined Macros 32

4. Overview of Structured-Descriptive Processing 33
4.1 Variable Names Available for the Structured Description 33
4.1.1 Register Variables 33
4.1.2 Flag Variables 34
4.1.3 A0 Register Bit Variables 34
4.1.4 Pointer Variables 35
4.2 Constants Available for the Structured Description 35
4.3 Operators Available for the Structured Description 36
4.3.1 Priority of Operators 38
4.4 Control Statements Available for the Structured Description 39
4.5 Bit-Manipulating Instructions 63
4.6 Concatenating Expressions Using Logical Operators 66
4.7 Automatic Generation of Constant Labels 69
4.8 Stack Areas Used for the Structured Description 71
4.9 Outputting the Structured Description to a List File 72

5. Overview of Assembling 74
5.1 Specifications of Conversion of Assembly Codes 74
5.2 Comments in Assembly Codes 82
5.3 Definitions of Data in Data Section 82
5.4 Pseudo-Directive in Assembly Codes 83
5.5 About Sections 85
5.5.1 Allocating Sections and the Number of Sections 86
5.5.2 Note on Defining Multiple Sections 88
5.6 Direct Description of Instruction Codes 88

6. Details of Preprocessing 89
6.1 Operators of Constant Expressions 89

7. Details of Structured-Descriptive Processing 91
7.1 Writing Address Values 91
7.2 Restrictions on the Structured Description 91
7.2.1 Expressions over Multiple Lines 91
7.2.2 Operators in Control Statements 91
7.2.3 Bit Manipulation Instructions 92

7.2.4 Variables That Cannot Be Handled by Operators 92
7.3 Crossing Nests in the Structured Description 94
7.4 Differences of Code Generation Depending on the Core Version of DSP 94
7.5 Character Sets Available in the Structured Description 95
7.6 Differences of Meanings at the Spots where "()" is Used 96
7.7 Note on Using the Structured Description when the V3 Core is Used 97
7.8 Note on Using the Structured Description without Side Effect 98

8. Details of Assembling 99
8.1 Restrictions on Assembling 99
8.2 Character Sets Available in the Assembly Description 99
8.3 Supplementary on Generating Assembly Codes 99

9. Reserved Words 101

10. Translation Limits 103
10.1 Translation Limits on Preprocessing 103
10.2 Translation Limits on the Structured Description 106
10.3 Translation Limit on Assembling 106

11. Error Messages 107
11.1 Formats of Error Messages 107
11.2 Error Messages 107

1. Overview

This document summarizes the functional specifications of the structured-descriptive assembler for FAA/GREEN DSP (DSPASM).

2. About DSPASM

The DSPASM is a program which assembles program codes having structured descriptions and outputs the assembled result as an object file.

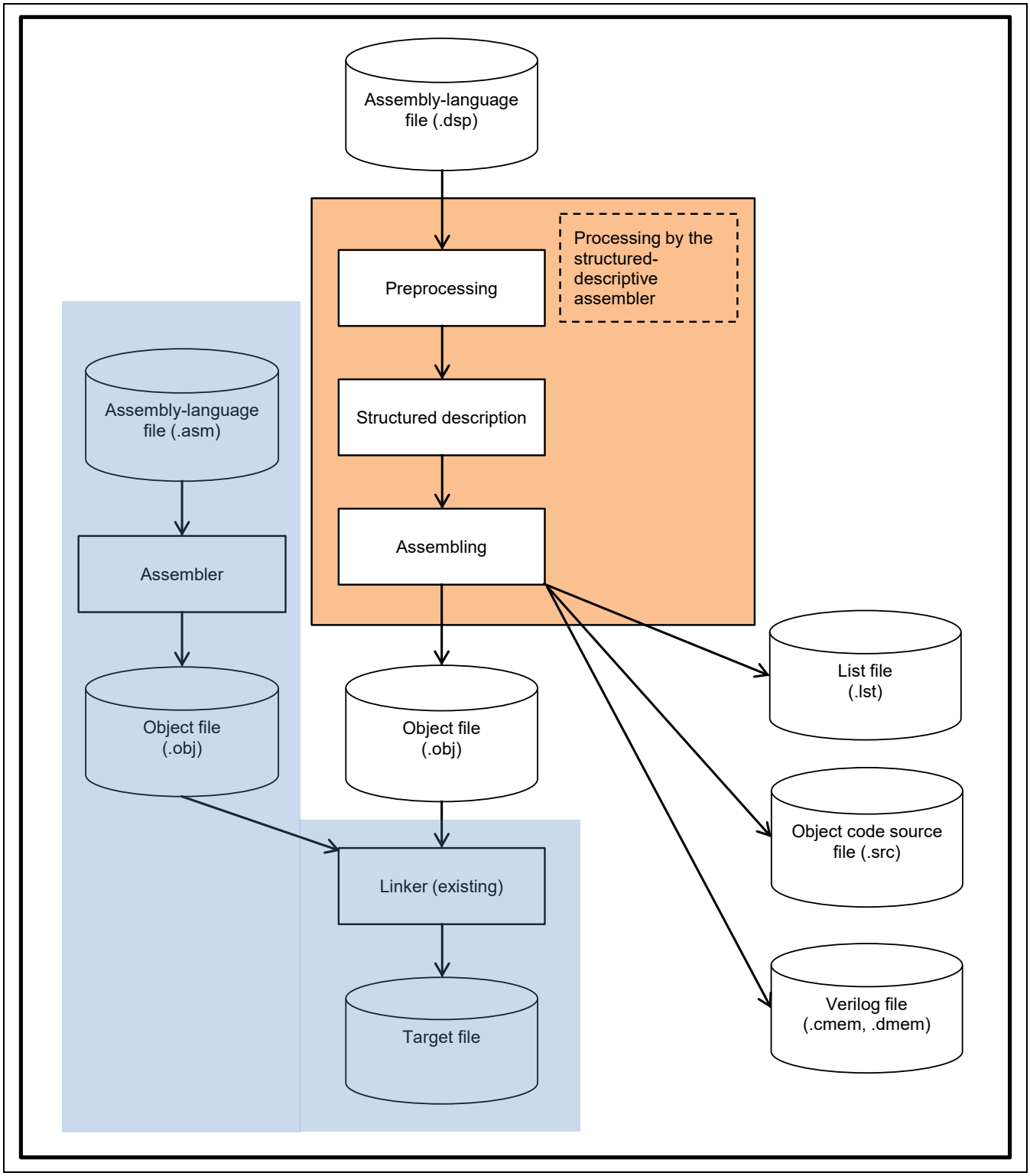

Figure 2.1 Flow of DSPASM Processing

2.1 Operating Environment

The DSPASM operates in the following environment.

Table 2.1 Operating Environment of the DSPASM

Microsoft Windows	Version 7 (32 bits or 64 bits, Japanese version or English version)
	Version 8 (32 bits or 64 bits, Japanese version or English version)
	Version 8.1 (32 bits or 64 bits, Japanese version or English version)
	Version 10 (32 bits or 64 bits, Japanese version or English version)
	Version 11 (64bits Japanese version or English version)

2.2 Input to the DSPASM

The DSPASM is input with the assembly-language file.

Source codes to be assembled are described in an assembly-language file. For source codes, the normal assembler description and the structured description can be used. In addition, preprocessing directives can be described in source codes.

The following shows an overview of an assembly-language file.

Table 2.2 Overview of an Assembly-Language File

Assembly-language file: extension (*.dsp)
Format
[Comment lines] [Preprocessing directives] SECTION CODE [NAME code section name] [LOCATE code RAM address] Program codes or comment lines SECTION DATA [NAME data section name] [LOCATE data RAM address] Data or comment lines
Sample code

```
; Sample code
;

#include <header.h>

SECTION CODE LOCATE H'd000
      MOV #S1_ST, DP0
      MOV #S1_OUT, RP0
      ;
      MOV (DP0+), A0                ; DataB->A0
      MOV A0, R0                    ; DataB->R0
      MOV (DP0+), A0                ; DataA->A0
      ADD                           ; DataA + DataB
      MOV A0, (RP0+)
      STOP
      ;
SECTION DATA LOCATE H'c000
S1_ST:   DATA H'00000050      ; DataB
         DATA H'00000100      ; DataA
         DATA H'00000003      ; Param_M0
;
S1_OUT: DATA H'00000000
```

2.3 Output from the DSPASM

One of the following three files can be output from the DSPASM.

1. Object code source file
2. Verilog file
3. Object file

The type of the file to be output is specified with the command-line option (-format).

For details on command-line options, refer to section 2.4, Command-Line Options for the DSPASM.

A list file can be output to check the result of conversion. When a list file is generated, the command-line option (-list) is specified.

The following shows overviews of each file.

(1) Object code source file

Table 2.3 Overview of an Object Code Source File

The assembled results are divided into program and data sections and output as two types of text files.

For the file that outputs the program section, "_dspcode.src" is added at the end of the file name.
For the file that outputs the data section, "_dspdata.src" is added at the end of the file name.

Output example of an object code source file (program: *_dspcode.src)

```
.SECTION            SAREA_DSPCODE, DATA, LOCATE=H'd000
 .ORG       H'd000
.DATA.B  H'80
.DATA.B             H'00
.DATA.B             H'88
.DATA.B             H'03
.DATA.B             H'07
.DATA.B             H'03
.DATA.B             H'07
.DATA.B             H'0F
.DATA.B             H'0D
.DATA.B             H'20
.END ND
```

Output example of an object code source file (data: *_dspdata.src)

```
.SECTION            SAREA_DSPDATA, DATA, LOCATE=H'c000
 .ORG       H'c000
S1_ST:
.DATA.B             H'00
.DATA.B             H'00
.DATA.B             H'00
.DATA.B             H'50
.DATA.B             H'00
.DATA.B             H'00
.DATA.B             H'01
.DATA.B             H'00
.DATA.B             H'00
.DATA.B             H'00
.DATA.B             H'00
.DATA.B             H'03
S1_OUT:
.DATA.B             H'00
.DATA.B             H'00
.DATA.B             H'00
.DATA.B             H'00
.END
```

(2) Verilog file

Table 2.4 Overview of a Verilog File

The assembled results are divided into program and data sections and output as two types of text files. For the file that outputs the program section, the extension of the file name is ".cmem". For the file that outputs the data section, the extension of the file name is ".dmem".
Output example of a Verilog file (program: *.cmem)
@00000000 80008803 @00000001 0703070f @00000002 0d200000
Output example of a Verilog file (data: *.dmem)
@00000000 00000050 @00000001 00000100 @00000002 00000003 @00000003 00000000

(3) Object file

Table 2.5 Overview of an Object File

The assembled results are output in the binary format (ELF/DWARF2). The extension of the file name is ".obj".

(4) List file

Table 2.6 Overview of a List File

The source code to be assembled and the assembled results are output to the same text file. The extension of the file name is ".lst".
List file (*.lst)
1 : .line "C:¥dspasm¥sample_code.dsp" , 1 2 : 3 : SECTION CODE 4 0000 8000 : MOV #S1_ST, DP0 5 0002 8803 : MOV #S1_OUT, RP0 6 : ; 7 : SECTION DATA 8 0000 00000050 : S1_ST: DATA H'00000050 9 0004 00000100 : DATA H'00000100 10 0008 00000003 : DATA H'00000003 11 : ; 12 000c 00000000 : S1_OUT: DATA H'00000000

2.4 Command-Line Options for the DSPASM

The DSPASM is executed in the following format with the command line of Windows ("Δ" means a space).

```
dspasmΔ[command-line option]Δan input source file name
```

Figure 2.2 Input Format of the DSPASM Command Line

An extension of an input source file name cannot be omitted. Only one file can be specified for the input source file name. If multiple input source files are specified, an assembler error will occur.

The following shows command-line options that can be specified with the DSPASM. When those options are specified, upper-case and lower-case characters are not distinguished.

Table 2.7 Command-Line Options

Command-Line Option	Function
-formatΔoutput format (output format: ASM/VERILOG/OBJ)	Specifies the output format for the assembler.
-list	Outputs a list file.
-outputΔfolder name	Specifies a folder in which files will be output.
-text_macroΔcharacter	Specifies the first character used to specify a text macro such as define or ifdef.
-defineΔname#value	Specifies a text macro.
-allow_text_macro_redefine	Allows redefining a symbol in the -define command-line option or the preprocessing directive in the source code.
-inc_dirΔfolder name	Specifies a source folder for a file to be included.
-dspΔDSP type (DSP type: RX_DSP / RL78_DSP / RL78_101_DSP / RL78_111_DSP / RL78_IAR_DSP / RL78_LLVM_DSP / RL78_GCC_DSP / ARM_DSP / ARM_EABI5_DSP)	Specifies DSP which is the target of code generation.
-core_versionΔversion (version: 2/3)	Specifies the core version of DSP which is the target of code generation.
-E	Preprocesses an input file and outputs the result to the file.
-cpuLittleEndian -cpuBigEndian	Selects the endian for the CPU.
-littleEndianData	Allows data values output as the result of assembling to be in little endian.
-code_section_start -data_section_start	Specifies addresses where sections of code or data are to start when allocated.
-no_debug_info	Information for use in debugging is not output in the object file.
-debug_aranges-no-padding	Specify this option when creating an input file for the converter (renesas_cc_converter).
-labelΔattribute (attribute: GLOBAL/LOCAL)	Specifies the attribute of the symbol.
-macro_identifyΔ identification method (identification method: FORWARD/EXACT)	Specifies the text macro identification method.
-dwarf_specΔoutput setting (output setting: INITIAL/GENERIC/RENESAS)	Specifies the contents of DWARF information to be output in an object file.
-code_execinstr	Set the SHF_EXECINSTR flag for the directive code section of an object file.

(1) Options for file outputs

Table 2.8 -format Command-Line Option

Specifying output files: -format	
Format	-formatΔoutput format Use any one of the following output formats: ASM: Outputs object code source files. VERILOG: Outputs Verilog files. OBJ: Outputs object files. Specifying the option without an output format leads to an assembler error.
Notes	Specifies the output format of the assembler. If this option is not specified, an assembler error will occur. If this option is specified several times, the last specification is effective.
Example	dspasm -format OBJ a.dsp

Table 2.9 -list Command-Line Option

Specifying the output of a list file: -list	
Format	-list
Notes	Outputs a list file.

Table 2.10 -output Command-Line Option

Specifying an output folder: -output	
Format	-outputΔfolder name Specifying the option without a folder name leads to an assembler error.
Notes	Specifies a folder in which files will be output. If this option is not specified, files are output to the folder in which dspasm.exe has been started from the command line. If this option is specified several times, the last specification is effective.
Example	dspasm -format OBJ -output .¥tmpdir a.dsp

(2) Options for preprocessing

Table 2.11 -text_macro Command-Line Option

Specifying the first character of a text macro: -text_macro	
Format	-text_macro∆character Use any one of the following characters: # ' ` @ _ Specifying the option without a character leads to an assembler error.
Notes	Specifies the first character used to specify a text macro such as define or ifdef. If a character other than above is specified, an assembler error will occur. If this option is not specified, # is regarded as being specified. If this option is specified several times, the last specification is effective.
Example	dspasm -format OBJ -text_macro @ a.dsp

Table 2.12 -define Command-Line Option

Specifying a text macro: -define	
Format	-define∆name#value Specifying the option without "name#value" leads to an assembler error.
Notes	Specifies a text macro. The character strings to be replaced are described before "#" and those that have been replaced are described after "#". When multiple text macros are specified, specify this option repeatedly by the required number of times. The character strings to be replaced and that have been replaced are forcedly processed as upper-case characters (upper-case and lower-case characters are not distinguished). Values after "#" can be omitted; in this case, spaces are inserted in the character strings that have been replaced. If two or more "#" are specified for the option, the character that has been described at the leftmost side is regarded as the delimiter.
Example	dspasm -format OBJ -define AAA#5 a.dsp

Table 2.13 -allow_text_macro_redefine Command-Line Option

Allowing redefinition of a text macro: -text_macro	
Format	-allow_text_macro_redefine

Notes	Allows redefining a symbol in the -define command-line option or the preprocessing directive in the source code.
	If this option is not specified, redefining a symbol is not allowed (an assembler error will occur).
	`#define REG_R R0`
	`#define REG_R R1 ; When the -allow_text_macro_redefine option is specified, the definition`
	`; of "define REG_R R1" that appeared later becomes enabled.`
	`; If this option is not specified, since redefinition of a symbol is not allowed,`
	`; an assembler error will occur.`
	`MOV A0,#REG_R ; When the -allow_text_macro_redefine option is specified, the character`
	`; string is replaced with "MOV A0,R1".`

Table 2.14 -inc_dir Command-Line Option

Specifying the include folder: -inc_dir	
Format	-inc_dir∆folder name Specifying the option without a folder name leads to an assembler error.
Notes	Specifies a source folder for a file to be included. If this option is not specified, a folder in which the assembly-language file has been stored is regarded as the starting point of the relative path. This option can be specified several times; in this case, a file to be included is searched in the order specified with the option.
Example	dspasm -format OBJ -inc_dir .¥abc a.dsp

(3) Options for code generation

Table 2.15 -dsp Command-Line Option

<table>
<tr><td colspan="2">Specifying DSP: -dsp</td></tr>
<tr><td>Format</td><td>-dsp∆DSP type
Use any one of the following DSP types:
<table>
<tr><th>Type specified with the option</th><th>Target chip</th></tr>
<tr><td>RX_DSP</td><td>RX</td></tr>
<tr><td>RL78_DSP</td><td>RL78 (conformant with earlier than CCRL V1.01)</td></tr>
<tr><td>RL78_101_DSP</td><td>RL78 (conformant with CCRL V1.01 or later)</td></tr>
<tr><td>RL78_111_DSP</td><td>RL78 (conformant with CCRL V1.11 or later)</td></tr>
<tr><td>RL78_IAR_DSP</td><td>RL78 (conformant with IAR tool for RL78)</td></tr>
<tr><td>RL78_LLVM_DSP</td><td>RL78 (conformant with LLVM for Renesas RL78)</td></tr>
<tr><td>RL78_GCC_DSP</td><td>RL78 (conformant with GCC for Renesas RL78)</td></tr>
<tr><td>ARM_DSP</td><td>ARM</td></tr>
<tr><td>ARM_EABI5_DSP</td><td>ARM (conformant with the Embedded Application Binary Interface (EABI) standard)</td></tr>
</table>
Specifying the option without a DSP type leads to an assembler error.</td></tr>
<tr><td>Notes</td><td>Specifies DSP which is the target of code generation.
If this option is not specified, "RX_DSP" is regarded as being specified.
If this option is specified several times, the last specification is effective.
"ARM_EABI5_DSP" is specified when a linkable object file is generated by the GNU or IAR tool for use with ARM cores.
To use an object file in a version earlier than CCRL V1.01, specify RL78_DSP.
To use an object file in CCRL V1.01 or later, specify RL78_101_DSP.
The code and data are handled as constants (CONSTF).
To use an object file in CCRL V1.11 or later, specify RL78_111_DSP.
The code and data are handled as those to be allocated to the DSP code and data areas in RL78 memory space.
To use an object file with the IAR tool for RL78, specify RL78_IAR_DSP.
To use an object file with LLVM for Renesas RL78, specify RL78_LLVM_DSP.
To use an object file with GCC for Renesas RL78, specify RL78_GCC_DSP.</td></tr>
<tr><td>Example</td><td>dspasm -format OBJ -dsp RL78_DSP a.dsp</td></tr>
</table>

Table 2.16 -core_version Command-Line Option

Specifying the core version of DSP: -core_version	
Format	-core_versionΔversion Use any one of the following versions: 2 The V2 core is the target of assembling. 3 The V3 core is the target of assembling. Specifying the option without a version leads to an assembler error.
Notes	Specifies the core version of DSP which is the target of code generation. If this option is not specified, version 3 is regarded as being specified. If this option is specified several times, the last specification is effective.
Example	dspasm -format OBJ -dsp RL78_DSP -core_version 2 a.dsp

Table 2.17 -E Command-Line Option

Outputting the result of preprocessing to a file: -E	
Format	-E
Notes	Preprocesses an input file and outputs the result to the file. For the file name to be output, the extension of the input file is changed to ".i". When this option is specified, the DSPASM only executes preprocessing; structured-descriptive processing and assembler execution are not performed. If this option is used together with -list, the list file will not be output.

Table 2.18 -cpuLittleEndian and-cpuBigEndian Command-Line Options

Selecting the endian for the CPU	
Format	-cpuLittleEndian -cpuBigEndian
Notes	Specify -cpuLittleEndian to select little endian for the CPU. If you specify -cpuBigEndian or none of these options, big endian will be selected for the CPU. If both the -cpuLittleEndian and -cpuBigEndian options are specified, the option specified later is valid. In cases where big endian is to be selected for the CPU, also specify -littleEndianData because data values in the program need to be inverted in 4-byte units.

Table 2.19 -littleEndianData Command-Line Option

Outputting data values in little endian: -littleEndianData	
Format	-littleEndianData
Notes	Specify this option if you wish data values output as the result of assembling to be in little endian. When this option is not specified, the output data values will be in big endian.

Table 2.20 -code_section_start Command-Line Option

<table>
<tr><td colspan="2">Specifying an address where a section of code is to start when allocated: -code_section_start</td></tr>
<tr><td>Format</td><td>-code_section_start∆ <first address of the section>

Specifying the option without a first address of the section leads to an assembler error.</td></tr>
<tr><td>Notes</td><td>Specify the address as a decimal or hexadecimal number.
For the hexadecimal notations, refer to Table 4.6, Notation Patterns for Hexadecimal Numbers.

The ranges of specifiable addresses for allocation are as follows. An assembler error occurs when a value which is out of the applicable range is specified.
<table>
<tr><td>-core_version 2 is specified.</td><td>0 to FFFh</td></tr>
<tr><td>-core_version 3 is specified, or -core_version is not specified.</td><td>0 to 3FFFh</td></tr>
</table>
The code section is allocated from address 0 when this option is not specified.</td></tr>
<tr><td>Example</td><td>dspasm -format OBJ -code_section_start 500h a.dsp</td></tr>
</table>

Table 2.21 data_section_start Command-Line Option

Specifying an address where a section of data is to start when allocated: -data_section_start	
Format	-data_section_startΔ<first address of the section> Specifying the option without a first address of the section leads to an assembler error.
Notes	Specify the address as a decimal or hexadecimal number. For the hexadecimal notations, refer to Table 4.6, Notation Patterns for Hexadecimal Numbers. The ranges of specifiable addresses for allocation are as follows. An assembler error occurs when a value which is out of the applicable range is specified. -core_version 2 is specified. — 0 to FFFh -core_version 3 is specified, or -core_version is not specified. — 0 to 1FFFh The data section is allocated from address 0 when this option is not specified.
Example	dspasm -format OBJ -data_section_start 300h a.dsp

Table 2.22 no_debug_info Command-Line Option

Disabling the output of information for use in debugging: -no_debug_info	
Format	-no_debug_info
Notes	Information for use in debugging is not output in the object file. When this option is not specified, information for use in debugging is output in the object file. Note, however, that the information is not output if the -dsp ARM_DSP option is specified, whether or not the -no_debug_info option is specified.

Table 2.23 label Command-Line Option

Specifies the attribute of the symbol: -label	
Format	-labeΔattribute
Notes	Specifies the attributes of symbols without a .public specification. Use any one of the following attributes: GLOBAL — The symbol can be referenced from other modules. LOCAL — The symbol can't be referenced from other modules. Specifying the option without a attribute leads to an assembler error. If this option is not specified, LOCAL is regarded as being specified. If this option is specified several times, the last specification is effective.

Table 2.24 macro_identify Command-Line Option

<table>
<tr><td colspan="2">Specifying the text macro identification method: -macro_identify</td></tr>
<tr><td>Format</td><td>-macro_identify∆identification method</td></tr>
<tr><td>Notes</td><td>Specifies the macro identification method when text macro replacement is performed.

Use any one of the following identification methods:

<table>
<tr><td>Identification method specified with the option</td><td>Description</td></tr>
<tr><td>FORWARD</td><td>A macro name character string after '#' is replaced with the macro definition.

Replacement is performed even if the identifier to be replaced is included in a part of other identifiers.</td></tr>
<tr><td>EXACT</td><td>A macro name character string is replaced with the macro definition only when the character string matches a token beginning with '#'. Other operations are the same as for FORWARD.</td></tr>
</table>

If this option is not specified, the macro name character string after '#' is replaced with the macro definition.

Replacement is performed even if the identifier to be replaced is included in the part of another identifier (FORWARD operation).

If this option is specified several times, the last specification is effective.</td></tr>
</table>

Table 2.25 dwarf_spec Command-Line Option

<table>
<tr><td colspan="2">Specifying DWARF information output specifications for object files: -dwarf_spec</td></tr>
<tr><td>Format</td><td>-dwarf_specΔoutput setting</td></tr>
<tr><td>Notes</td><td>Specifies the specifications of DWARF information to be output in an object file.

Use any one of the following output settings:

<table>
<tr><th>Output setting specified with the option</th><th>Description</th></tr>
<tr><td>INITIAL</td><td>Outputs information according to the DWARF specifications for DSPASM.</td></tr>
<tr><td>GENERIC</td><td>Outputs information according to the DWARF specifications for GDB.</td></tr>
<tr><td>RENESAS</td><td>Outputs information according to the DWARF specifications for CCRL/CCRX.</td></tr>
</table>

If this option is not specified, INITIAL setting is assumed.

If this option is specified several times or at the same time as the -debug_aranges-no-padding option, the last specified option has priority.</td></tr>
</table>

Table 2.26 code_execinstr Command-Line Option

<table>
<tr><td colspan="2">Enabling the SHF_EXECINSTR flag setting for code section: -code_execinstr</td></tr>
<tr><td>Format</td><td>-code_execinstr</td></tr>
<tr><td>Notes</td><td>Sets the SHF_EXECINSTR flag for the code section in an object file.

If this option is not specified and one of the following is specified for the -dsp option, the SHF_EXECINSTR flag is not set:

ARM_DSP, ARM_EABI5_DSP, RL78_IAR_DSP, RL78_101_DSP, or RL78_111_DSP.

If this option is not specified and a value other than the following is specified for the -dsp option, the SHF_EXECINSTR flag is set:

ARM_DSP, ARM_EABI5_DSP, RL78_IAR_DSP, RL78_101_DSP, or RL78_111_DSP.</td></tr>
</table>

3. Overview of Preprocessing

In the DSPASM, the following preprocessing is performed for the assembly-language file to be input.

- Macro replacement (replacing a character string described in a file)
- Conditional inclusion (selecting a code block to be assembled)
- File inclusion (including an external file)

This chapter describes the overview of preprocessing.

For details on preprocessing, refer to chapter 6, Details of Preprocessing.

3.1 Distinguishing Identifiers for Preprocessing

In the DSPASM, a specific character (text macro character) is used to determine whether the contents described in the assembly-language file are the preprocessing directive or the character strings for macro replacement.

The text-macro character can be specified with the "-text_macro" command-line option and "#" is used by default.
For characters that can be used for the argument of the option, refer to Table 2.11.

Since a character can be specified for the text-macro character, this chapter uses "{TC}" as the text-macro character.

3.2 Macro Replacement

In the DSPASM, a specific character string described in the assembly-language file can be replaced with another character string, which is called as macro replacement.

Macro replacement targets the character string beginning with {TC}. The "{TC} define" preprocessing directive or the "-define" command-line option is used to determine what replacement is performed.

Table 3.1 define Preprocessing Directive

Macro replacement preprocessing directive: {TC}define
Format: 1) {TC}define∆identifier∆list of replacement elements 2) {TC}define∆identifier([list of identifiers])∆list of replacement elements 3) {TC}define∆identifier(…)∆list of replacement elements 4) {TC}define∆identifier(list of identifiers, …)∆list of replacement elements The first character of the formats above begins with {TC}.
Sample code (This sample code uses "#" instead of {TC}.)

```
; Sample code
;
#define MAGIC_NO     #H'123
mov #MAGIC_NO, DP0                    ; "#MAGIC_NO" is replaced with "H'123".

#define CLEAR_A0()   A0=0             ; Function-macro replacement
#CLEAR_A0()                           ; The result is replaced with "A0 = 0".

#define SET_A0(VAL)   A0 = #VAL       ; Function-macro replacement with arguments
#SET_A0(1)                            ; The result is replaced with "A0 = 1".

#define SET_REG(…)   mov   #__VA_ARGS__        ; Function-macro replacement using variable arguments
                                               ; The content of an argument is reflected to the predefined macro
                                               ; __VA_ARGS__.
#SET_REG(#H'123, DP0)                          ; The result is replaced with "mov H'123, DP0".
                                               ; Multiple arguments can be specified.

#define SET_REG2(R, …)       mov   #__VA_ARGS__, #R   ; Function-macro replacement using variable arguments
                                                      ; The first argument is reflected to R and the contents of other
arguments
                                                      ; are reflected to the predefined macro __VA_ARGS__.
#SET_REG2(DP0, #H'ABC)                                ; The result is replaced with "mov H'ABC, DP0".
```

Notes:

a) For symbols for macro replacement, upper-case and lower-case characters are not distinguished.

b) Arguments specified with variable arguments are expanded to the predefined macro "__VA_ARGS__".

c) Macro replacement is performed even if an identifier to be replaced is included in a part of other identifiers. To perform replacement in units of tokens, specify the command-line option "-macro_identify EXACT". In addition, identifiers that have previously been defined are preferentially replaced.

```
#define     VALUE    8
MOV #H'7#VALUE9, DP0                    ; The result is replaced with " MOV #H'789, DP0".

#define     AA       #H'1
#define     AAA      #H'9
MOV #AA23, DP0X                                 ; The result is replaced with "#H'123".
```

d) Macros are replaced several times on the same line (replacement can be nested).

```
#define    VAL_TYPE    #ZERO
#define    ZERO        H'00000000
A0 = #VAL_TYPE                          ; A0 is replaced with H'00000000.
```

e) When replacement is performed several times on the same line, the replaced result depends on the order that the text macro has been defined.

```
#define    ZERO        H'00000000
#define     VAL_TYPE    #ZERO
A0 = #VAL_TYPE                          ; A0 is replaced with #ZERO.
```

f) Macros are replaced even if the identifier to be replaced is the reserved word of the DSPASM.

g) Macro replacement cannot be redefined. When redefinition is allowed, specify the command-line option "-allow_text_macro_redefine". When "-allow_text_macro _redefine" is specified, the definition of macro replacement which has been specified with the command-line option "-define" can be overwritten by the {TC}define directive in the source code.

h) The preprocessing directive for macro replacement cannot be described over multiple lines.

```
#define    MANY_ARGS(arg1, arg2, arg3,¥     ; If the definition is described over multiple lines, an assembler error will
arg4, arg5, arg6)   VAL                     ; occur.
```

Maximum values of the character string length and the number of identifiers:

For the maximum values of the character string length and the number of identifiers, refer to section 10.1, Translation Limits on Preprocessing.

Error descriptions:

◊ In the macro replacement preprocessing directive, lists of replacement elements can be omitted.

◊ In the function-macro definition without variable arguments, descriptions to be replaced and the number of arguments for the macro definition must be matched.

```
#define    CLEAR_A0()          A0 = 0
#CLEAR_A0(1)                   ; An assembler error will occur because the number of arguments is not matched.

#define    SET_A0(VAL)         A0 = #VAL
#SET_A0()                      ; An assembler error will occur because the number of arguments is not matched.
```

◊ An assembler error will occur if an identifier which is used for the function-macro definition is described in the format without "()".

```
#define      CLEAR_A0()        A0 = 0
#CLEAR_A0                      ; An assembler error will occur because the identifier does not have "()".
```

◊ Argument names for the function-macro definition can consist only of alphabets, numerals, and underscores ("_").

```
#define   CLEAR_A0(ARG+1)   A0 = #ARG+1     ; An assembler error will occur because the argument name has symbols.
#define   CLEAR_A0(ARG@1)   A0 = #ARG@1     ; An assembler error will occur because the argument name has symbols.
```

3.3 Conditional Inclusion

In the DSPASM, it can be determined whether or not a part of source codes described in the assembly-language file is the target of assembling according to a specific condition, which is called as conditional inclusion.

The following two types of conditions are determined in conditional inclusion.

1) Is a specific symbol defined?
2) Is a constant expression satisfied?

The preprocessing directive "{TC}ifdef" or "{TC}ifndef" is used to determine whether or not there is a symbol definition. When the result of a constant expression is determined, the preprocessing directive "{TC}if" is used.

Table 3.2 if Preprocessing Directive

Conditional inclusion preprocessing directive according to the result of a constant expression: {TC}if

Format:

1) {TC}if∆constant expression

Description block of the next source code to be included

{TC}elif∆constant expression

Description block of the next source code to be included

{TC}else

Description block of the next source code to be included

{TC}endif

The first character of the formats above begins with {TC}.

It is possible to omit {TC}elif and {TC}else.

Some {TC}elif can be described in a block between {TC}if and {TC}endif.

Sample code (This sample code uses "#" instead of {TC}.)

```
; Sample code
;
SECTION CODE

#define NUM_UNITS     2
#if       #NUM_UNITS >= 2
    ;
    ; Processing when NUM_UNITS is 2 or more
    ;
#elif   #NUM_UNITS == 1
    ;
    ; Processing when NUM_UNITS is 1
    ;
#else
    ;
    ; Processing when NUM_UNITS is neither 2 or more nor 1
    ;
#endif
```

Notes:

a) For operators that can be used for constant expressions, refer to chapter 6, Details of Preprocessing.

b) Constant expressions cannot be omitted.

c) Constant expressions cannot use assignment operators, unary additive operators, and unary subtractive operators.

d) The priority of operation in a constant expression can be changed by using "()".

e) Constant expressions support decimal and hexadecimal descriptions.

The length of a constant expression and the maximum value of nesting counts:

For the length of a constant expression and the maximum value of nesting counts, refer to section 10.1, Translation Limits on Preprocessing.

Error descriptions:

◊ {TC}endif corresponding to {TC}if must be described in the same files.

◊ In a constant expression, an assembler error will occur if the format of the expression is invalid; e.g., an expression on the right side or the left side of a binary operator is not described or parentheses are not matched.

◊ An assembler error will occur if division by 0 or residue operation by 0 is generated during processing of constant expressions.

Table 3.3 ifdef/ifndef Preprocessing Directive

Conditional inclusion preprocessing directive according to the symbol definition: {TC}ifdef/{TC}ifndef
Format:
1) {TC}ifdef∆identifier *Description block of the next source code to be included* {TC}else *Description block of the next source code to be included* {TC}endif 2) {TC}ifndef∆identifier *Description block of the next source code to be included* {TC}else *Description block of the next source code to be included* {TC}endif The first character of the formats above begins with {TC}. It is possible to omit {TC}else. {TC}elif can be described in a block between {TC}if and {TC}endif.
Sample code (This sample code uses "#" instead of {TC}.)

```
; Sample code
;
SECTION CODE

#define TEST_VERSION
#ifdef   TEST_VERSION    ;
    ; Processing when TEST_VERSION is defined
    ;
#else
    ;
    ; Processing when TEST_VERSION is not defined
    ;
#endif

#define   CUSTOM_ONLY
#ifndef   CUSTOM_ONLY
    ;
    ; Processing when CUSTOM_ONLY is not defined
    ;
#else
    ;
    ; Processing when CUSTOM_ONLY is defined
    ;
#endif

#define VERSION__NUM    10
#ifdef   VERSION__NUM
    ;
    ; Regardless how VERSION__NUM is replaced,
    ; this block is the target of assembling if a symbol has been defined.
    ;
#endif
```

Notes:

a) Preprocessing directives, {TC}ifdef and {TC}ifndef, only determine whether or not there is a symbol definition regardless of the value that the symbol is replaced. When conditions are determined according to the value of the symbol, use the preprocessing directive {TC}if.

b) For identifiers, upper-case and lower-case characters are not distinguished.

c) Identifiers cannot be omitted.

The maximum value of nesting counts:

For the maximum value of nesting counts, refer to section 10.1, Translation Limits on Preprocessing.

Error description:

◊ {TC}endif corresponding to {TC}ifdef or {TC}ifndef must be described in the same files.

3.4 File Inclusion

In the DSPASM, the contents of other files can be included in the assembly-language file which is the target of assembling, which is called as file inclusion.

When files are included, the preprocessing directive "{TC}include" is used.

Table 3.4 include Preprocessing Directive

File inclusion preprocessing directive: {TC}include
Format:
1) {TC}include∆<file name> The first character of the format above begins with {TC}. It is possible to omit {TC}elif and {TC}else. Some {TC}elif can be described in a block between {TC}if and {TC}endif.
Sample code (This sample code uses "#" instead of {TC}.)
; Sample code ; SECTION CODE #include <value_no.def> ; The value_no.def file is included. #include <..\def\const.def> ; A file specified with the relative path is included. #define STARTUP_FILE startup.asm ; The file name is defined as a symbol. #include <#STARTUP_FILE> ; The "startup.asm" file is included.
Notes: a) A path name can be added to a file name. Both absolute and relative paths are supported as the path names. b) Both characters "\" and "/" are supported as delimiters of the path names. c) A source folder for a relative path can be specified with the command-line option "-inc_dir". If the "-inc_dir" option is not specified, a folder storing the assembly-language file is regarded as the source folder for the relative path. d) Symbols "<" and ">" are only supported for enclosing a file name. e) When the macro-replacement preprocessing directive {TC}define is described in a file that has been included by {TC}include, the symbol definition becomes valid from the line that {TC}define is described.
The maximum value of nesting counts for file inclusion:
For the maximum value of nesting counts for file inclusion, refer to section 10.1, Translation Limits on Preprocessing.
Error description:
◊ An assembler error will occur if no file can be read.

3.5 Predefined Macros

In the DSPASM, the following symbols are used as predefined macros.

These symbols are reserved words; the user cannot change the contents of definition.

- __VA_ARGS__
- __RENESAS__
- __RENESAS_VERSION__
- __DSPASM__

4. Overview of Structured-Descriptive Processing

When source code is described in the DSPASM, structured descriptions specific to the DSPASM can be used in addition to the assembler description.

Structured descriptions of the DSPASM support the creation of user programs and provide the following features.

- Program structures such as a branch, multiple branches, and iteration can be described in a format which is close to the C language.
- Numeric operations, assignments, bitwise operations, and comparison manipulations for memory or registers can be written in a format which is close to the C language.
- Statements for bitwise operations and bit comparison manipulations can be simpler than a normal assembly description.

This chapter describes the overview of structured-descriptive processing.

For details on structured-descriptive processing, refer to chapter 7, Details of Structured-Descriptive Processing.

4.1 Variable Names Available for the Structured Description

The following variable names can be used for the structured description.

- Register variables
- Flag variables
- A0 register bit variables
- Pointer variables

4.1.1 Register Variables

The following register variables can be used for the structured description.

Table 4.1 Register Variables

Variable Name	Register Type
A0	Accumulator register
M0	Multiplier register
M1	Shift register
L0	Upper-limit register
L1	Lower-limit register
R0	Adder register
R1	Adder register 1
DP0	Address pointer for accumulators
DP1	Address pointer for operation parameters
RP0	Address pointer for storing operation results
F0	Flag register
PS0	Program segment register
DS0	Data segment register
SS0	Stack segment register

Note: R1, F0, PS0, DS0, and SS0 register variables are only supported when "3" is specified for the argument of command-line option "-core_version".

Note that the BR0 register and PG0 are not supported in the structured description because assembler directives which operate registers do not exist. In addition, the SP0 register is not supported in the structured description because a stack is used in the assembly code generated in the structured description and SP0 is rewritten as a result.

When those registers are operated, the user must describe the ordinary assembly code.

4.1.2 Flag Variables

The following flag variables can be used for the structured description.

Table 4.2 Flag Variables

Variable Name	Register Type
I	Interrupt flag
Z	ZERO flag
U	UNDER flag
O	OVER flag

4.1.3 A0 Register Bit Variables

In the structured description, using register bit variables enables direct operation of bits in the A0 register.

The specified bit must be within the range from 0 to 31. If a bit outside the range is specified, an assembler error will occur.

Table 4.3 A0 Register Bit Variables

Variable Name	Register Type
A0_0	The least significant bit in the A0 register
A0_1	The second bit from the lower-order bit side in the A0 register
A0_2	The third bit from the lower-order bit side in the A0 register
Omitted	
A0_29	The third bit from the higher-order bit side in the A0 register
A0_30	The second bit from the higher-order bit side in the A0 register
A0_31	The most significant bit in the A0 register

The following shows an image of A0 register bit variables.

Table 4.4 Image of A0 Register Bits

Higher-order bit side ← A0 register → Lower-order bit side						
31	30	29	Omitted	2	1	0

4.1.4 Pointer Variables

Table 4.5 Pointer Variables

Variable Name	Register Type
(DP0)	Address pointer for accumulators
(DP1)	Address pointer for operation parameters
(*XX*)	Memory for data
(*NN* DP1)	Address pointer for operation parameters
(*NN*, DP0)	Address pointer for accumulators
(*NN*, RP0)	Address pointer for storing operation results
(DP0+R0)	Address pointer for accumulators + adder register

Notes: 1. Immediates or label names can be written to "*XX*" in the table.
2 Immediates from -128 to 127 can be written to "*NN*" in the table.
3. Note that "()" in an expression in a control statement (for details, refer to section 4.4, Control Statements Available for the Structured Description) does not mean a pointer variable but the change of the priority of operation. For details, refer to section 7.6, Differences of Meanings at the Spots where "()" is Used .
4. Pointer variable (DP0+R0) is only supported when "3" is specified for the argument of command-line option "-core_version".

4.2 Constants Available for the Structured Description

The structured description supports decimal and hexadecimal constants.

Although there are no limitations on the number of digits in decimal and hexadecimal numbers, an assembler error will occur if a value exceeds four bytes.

For the hexadecimal constants, any of the following patterns can be used.

Table 4.6 Notation Patterns for Hexadecimal Numbers

Notation Pattern	Notes
H'xxxxxxxx	Prefix H is appended to the hexadecimal constant.
h'xxxxxxxx	The prefix can be either upper- or lower-case.
xxxxxxxxh	Suffix H is appended to the hexadecimal constant.
xxxxxxxxH	The suffix can be either upper- or lower-case.

Note: "x" in the table means a hexadecimal digit (the characters A to F or a to f can be used in addition to 0 to 9).

4.3 Operators Available for the Structured Description

Table 4.7 Operators

Type	Operator	Description
Unary operators	+	Indicates positive numbers.
	-	Indicates negative numbers.
	~	Performs bitwise-inversion operation (NOT).
	++	Performs addition (unsigned).
	++.s	Performs addition (signed, saturation operation)
	--	Performs subtraction (unsigned).
	--.s	Performs subtraction (signed, saturation operation)
Division and multiplication operators	*	Performs multiplication (unsigned).
	*.s	Performs multiplication (signed, saturation operation)
	/	Performs division (unsigned). This instruction cannot be used for the V2 core; if it is used, an assembler error will occur.
	%	Performs residue operation (unsigned). This instruction cannot be used for the V2 core; if it is used, an assembler error will occur.
Addition and subtraction operators	+	Performs addition (unsigned).
	+.s	Performs addition (signed, saturation operation)
	-	Performs subtraction (unsigned).
	-.s	Performs subtraction (signed, saturation operation)
Bitwise shift operators	<<	Performs logical left-shift operation for the specified bits.
	<<.s	Performs arithmetic left-shift operation for the specified bits.
	>>	Performs logical right-shift operation for the specified bits.
	>>.s	Performs arithmetic right-shift operation for the specified bits.
Relational operators	<	The operation result is true when the left side of the expression is smaller than the right side (unsigned). This instruction cannot be used for the V2 core; if it is used, an assembler error will occur.
	<.s	The operation result is true when the left side of the expression is smaller than the right side (signed).
	>	The operation result is true when the left side of the expression is larger than the right side (unsigned). This instruction cannot be used for the V2 core; if it is used, an assembler error will occur.
	>.s	The operation result is true when the left side of the expression is larger than the right side (signed).
	<=	The operation result is true when the left side of the expression is smaller than or equal to the right side (unsigned). This instruction cannot be used for the V2 core; if it is used, an assembler error will occur.
	<=.s	The operation result is true when the left side of the expression is smaller than or equal to the right side (signed).
	>=	The operation result is true when the left side of the expression is larger than or equal to the right side (unsigned). This instruction cannot be used for the V2 core; if it is used, an assembler error will occur.

	>=.s	The operation result is true when the left side of the expression is larger than or equal to the right side (signed).
Equality operators	==	The operation result is true when the left side of the expression is equal to the right side.
	!=	The operation result is true when the left side of the expression is not equal to the right side.
Bitwise AND/OR/exclusive-OR operators	&	Performs bitwise AND operation (AND).
	|	Performs bitwise OR operation (OR).
	^	Performs bitwise exclusive-OR operation (XOR).
Logical AND/OR operators	&&	The operation result is true when the values of the left and right sides of the expression are compared to 0 and both of them are not equal. Short-circuit evaluation is performed (if the left side is false, the right side is not evaluated).
	||	The operation result is true when the values of the left and right sides of the expression are compared to 0 and either of them is not equal. Short-circuit evaluation is performed (if the left side is true, the right side is not evaluated).
Assignment operators	=	The right side is assigned to the left side. var ← exp
	*=	The result of multiplication between the left and right sides is assigned to the left side (unsigned). var ← var * exp
	*=.s	The result of multiplication between the left and right sides is assigned to the left side (signed, saturation operation). var ← var * exp
	/=	The result of division between the left and right sides is assigned to the left side (unsigned). var ← var / exp This instruction cannot be used for the V2 core; if it is used, an assembler error will occur.
	%=	The result of residue operation between the left and right sides is assigned to the left side (unsigned). var ← var % exp This instruction cannot be used for the V2 core; if it is used, an assembler error will occur.
	+=	The result of addition between the left and right sides is assigned to the left side (unsigned). var ← var + exp
	+=.s	The result of addition between the left and right sides is assigned to the left side (signed, saturation operation). var ← var + exp
	-=	The result of subtraction between the left and right sides is assigned to the left side (unsigned). var ← var - exp
	-=.s	The result of subtraction between the left and right sides is assigned to the left side (signed, saturation operation). var ← var - exp
	<<=	The result of logical left-shift operation for the right-side bits in the left side is assigned to the left side. var ← var << exp

	<<=.s	The result of arithmetic left-shift operation for the right-side bits in the left side is assigned to the left side. var ← var << exp
	>>=	The result of logical right-shift operation for the right-side bits in the left side is assigned to the left side. var ← var >> exp
	>>=.s	The result of arithmetic right-shift operation for the right-side bits in the left side is assigned to the left side. var ← var >> exp
	&=	The result of AND operation for the left and right sides is assigned to the left side. var ← var and exp
	|=	The result of OR operation for the left and right sides is assigned to the left side. var ← var or exp
	^=	The result of exclusive-OR operation for the left and right sides is assigned to the left side. var ← var xor exp
	Compound assignment (=..=)	The rightmost-side value is assigned to the left side. var ← var2 ← …← exp

Notes: 1. In the table above, 'var' and 'var2' mean variables and 'exp' means any expression (an expression consists of the combination of variables, constants, and operators).
2. Unary operators '++' and '--' are only supported when they are described in the prefix notation. If they are described in the postfix notation such as "A0—" or "A0++", an assembler error will occur.
3. If the specified variable cannot be handled by the operator, an assembler error will occur. (For details, see section 7.2.4, Variables That Cannot Be Handled by Operators.)

4.3.1 Priority of Operators

The following shows the priority of operators that are used for the structured description.

Table 4.8 Priority of Operators

High	(1)	+, - , ~, ++, ++.s, --, --.s (unary operators)
↑	(2)	*, *.s, /, %, <<, <<.s, >>, >>.s
	(3)	+. +.s, -, -.s
	(4)	&, | , ^
	(5)	==, !=, <, <.s, >, >.s, <=, <=.s, >=, >=.s
↓	(6)	&&, ||
Low	(7)	=, *=, *=.s, /=, %=, +=, +=.s, -=, -=.s, <<=, <<=.s, >>=, >>=.s, &=, |=, ^=

For an expression in a control statement (for details, refer to section 4.4, Control Statements Available for the Structured Description), when the expression is enclosed by "()", the execution priority of operation can be changed. The expression enclosed by "()" has the higher priority of operation than other expressions.

Note that "()" means the pointer variable in other expressions in the control statement; it cannot be used to change the execution priority of operation.

4.4 Control Statements Available for the Structured Description

The following control statements are available for the structured description.

- if … elif … else … endif
- switch … case … endsw
- while … endwh
- do … during
- for … to … step … endfor
- goto
- continue
- break

Table 4.9 if Control Statement

Conditional branch: if … elif … else … endif	

Format:
if∆[expression 1]
statement 1
elif∆[expression 2]
statement 2
else
statement3
endif
 	
*Some 'elif' can be described in an if statement.	

Function: The if statements are used to branch the control of a program according to the true or false value of the result when an expression has been evaluated. 1) When an evaluation result of the expression described with 'if' or 'elif' is true, the control of a program is resumed at the corresponding statements. 2) When all evaluation results of the expression described with 'if' or 'elif' are false, the control of a program is resumed at the statements described with 'else'.	
Specifications: a) Variables, constants, and operators can be described in expressions. b) The following operators are available for expressions. • Relational operators (<, >, <=, >=, <.s, >.s, <=.s, >=.s) • Equality operators (==, !=) • Logical AND/OR operators (&&, ||) • Bitwise AND/OR/exclusive-OR operators (&, |, ^) c) The description of 'elif' or 'else' can be omitted. d) Some 'elif' can be described in an if statement. e) 'endif' cannot be omitted.	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)

if [A0 < R0]
R0 = A0
else
A0 = R0
endif
	;--- Saves registers. ---
	PUSH A0
	PUSH RP0
	;--- Acquires operand 2. ---
	PUSH RP0
	PUSH A0
	PUSH R0
	POP A0
	MOV.L #__V_00000002,RP0
	MOV A0,(RP0+)
	POP A0
	POP RP0
	;--- Acquires operand 1. ---

```
                PUSH A0
                POP A0
                ;--- Executes comparison. ---
                PUSH A0
                PUSH L0
                PUSH L1
                PUSH DP1
                MOV A0,L0
                MOV.L #__V_00000002,DP1
                MOV (DP1+),L1
                CLAMP UNSIGNED
                POP DP1
                POP L1
                POP L0
                POP A0
               ;--- Stores the result of comparison. ---
                MOV.L (#__C_00000001),A0
                MOV.L #__V_00000001,RP0
                MOV A0,(RP0+)
                 ; --- Restores registers. ---
                POP RP0
                POP A0
                ; --- Performs conditional branches. ---
                JMP UNDER, #__L_00000004
                ;--- Stores the result of comparison. ---
                PUSH A0
                PUSH RP0
                MOV.L (#__C_00000000),A0
                MOV.L #__V_00000001,RP0
                MOV A0,(RP0+)
                POP RP0
                POP A0
                 ; --- Performs conditional branches.
                JMP #__L_00000002
              __L_00000004:
                PUSH   A0
                POP    R0
                PUSH   A0
                PUSH   RP0
                PUSH A0
                POP    A0
                MOV.L              #__V_00000001,RP0
                MOV   A0,(RP0+)
                POP    RP0
                POP    A0
                PUSH   A0
```

```
                PUSH   RP0
                MOV.L   (#__V_00000001),A0
                MOV.L   #__V_00000002,RP0
                MOV    A0,(RP0+)
                MOV.L   #__V_00000003,RP0
                MOV   A0,(RP0+)
                POP    RP0
                POP    A0
                JMP #__L_00000003
            __L_00000005:
                 PUSH  R0
                POP    A0
                PUSH              A0
                PUSH              RP0
                PUSH   R0
                POP    A0
                MOV.L   #__V_00000001,RP0
                MOV   A0,(RP0+)
                POP    RP0
                POP    A0
                PUSH   A0
                PUSH   RP0
                MOV.L   (#__V_00000001),A0
                MOV.L   #__V_00000002,RP0
                MOV   A0,(RP0+)
                MOV.L   #__V_00000003,RP0
                MOV   A0,(RP0+)
                POP    RP0
                POP    A0
                JMP #__L_00000003
             __L_00000003:

SECTION DATA NAME SAREA_DSPDATA
__C_00000000:          DATA H'00000000
__C_00000001:          DATA H'00000001
__V_00000001:          DATA H'00000000
__V_00000002:          DATA H'00000000
__V_00000003:          DATA H'00000000
```

Table 4.10 switch … case Control Statement

Multiple branches: switch … case … endsw

Format:

```
switch∆[expression]
case∆value:
   statement 1
   break
case∆value:
   statement 2
   break
default:
   statement 3
   break
endsw
```

Function:

The switch statements are used to branch the control of a program according to the value of an expression.

1) The control of a program is resumed at the statement that the value of the expression matches the description of the case label.

2) If the value of the expression does not match the description of the case label, the control of a program is resumed at the default statement.

3) If the value of the expression does not match the description of the case label and the default clause has not been described, any statements will not be executed.

Specifications:

a) Variables, constants, and unary operators can only be described in expressions.

b) Constants can only be described as the values of the case label.

c) Two or more case labels having the same value cannot be written. If the value of a case label is duplicated, an assembler error will occur.

d) The description of 'break' can be omitted; if it is omitted, processing described immediately after the next case label or the default label will be executed.

e) The description of 'default' can be omitted.

f) 'endsw' cannot be omitted.

Example of expansion of structured descriptions:

Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)
switch [A0]	__L_00000001:
case 1:	PUSH A0
...	PUSH RP0
break	PUSH RP0
case 2:	PUSH A0
...	PUSH A0
break	POP A0
case 3:	MOV.L #__V_00000002,RP0
...	MOV A0,(RP0+)
break	POP A0
default:	POP RP0
...	MOV.L (#__C_00000000),A0

break	PUSH DP0
endsw	MOV.L #__V_00000002,DP0
	CMP (DP0+)
	POP DP0
	MOV.L (#__C_00000001),A0
	MOV.L #__V_00000001,RP0
	MOV A0,(RP0+)
	POP RP0
	POP A0
	JMP NOT_ZERO, #__L_00000006
	PUSH A0
	PUSH RP0
	MOV.L (#__C_00000000),A0
	MOV.L #__V_00000001,RP0
	MOV A0,(RP0+)
	POP RP0
	POP A0
	JMP #__L_00000003
	__L_00000006:
	__L_00000003:
	; case 1:
	; --- Saves registers. ---
	PUSH A0
	PUSH RP0
	; --- Acquires operand 2. ---
	PUSH RP0
	PUSH A0
	MOV.L #__V_00000004,RP0
	MOV.L (#__C_00000001),A0
	MOV A0,(RP0+)
	POP A0
	POP RP0
	; --- Acquires operand 1. ---
	PUSH A0
	POP A0
	; --- Executes comparison. ---
	PUSH DP0
	MOV.L #__V_00000004,DP0
	CMP (DP0+)
	POP DP0
	MOV.L (#__C_00000001),A0
	MOV.L #__V_00000001,RP0
	MOV A0,(RP0+)
	; --- Restores registers. ---
	POP RP0
	POP A0

```
        ; --- Performs conditional branches.
        JMP ZERO, #__L_00000009
        PUSH A0
        PUSH RP0
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_0000000a
__L_00000009:
        PUSH A0
        PUSH RP0
        MOV.L (#__V_00000001),A0
        MOV.L #__V_00000002,RP0
        MOV A0,(RP0+)
        MOV.L #__V_00000003,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_00000005
__L_0000000a:
        ; case 2:
        ; --- Saves registers. ---
        PUSH A0
        PUSH RP0
        ; --- Acquires operand 2. ---
        PUSH RP0
        PUSH A0
        MOV.L #__V_00000004,RP0
        MOV.L (#__C_00000002),A0
        MOV A0,(RP0+)
        POP A0
        POP RP0
        ; --- Acquires operand 1. ---
        PUSH A0
        POP A0
        ; --- Executes comparison. ---
        PUSH DP0
        MOV.L #__V_00000004,DP0
        CMP (DP0+)
        POP DP0
        MOV.L (#__C_00000001),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        ; --- Restores registers. ---
```

```
        POP RP0
        POP A0
        ; --- Performs conditional branches.
        JMP ZERO, #__L_0000000d
        PUSH A0
        PUSH RP0
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_0000000e
__L_0000000d:
        PUSH A0
        PUSH RP0
        MOV.L (#__V_00000001),A0
        MOV.L #__V_00000002,RP0
        MOV A0,(RP0+)
        MOV.L #__V_00000003,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_00000005
__L_0000000e:
        ; case 3:
        ; --- Saves registers. ---
        PUSH A0
        PUSH RP0
        ; --- Acquires operand 2. ---
        PUSH RP0
        PUSH A0
        MOV.L #__V_00000004,RP0
        MOV.L (#__C_00000003),A0
        MOV A0,(RP0+)
        POP A0
        POP RP0
        ; --- Acquires operand 1. ---
        PUSH A0
        POP A0
        ; --- Executes comparison. ---
        PUSH DP0
        MOV.L #__V_00000004,DP0
        CMP (DP0+)
        POP DP0
        MOV.L (#__C_00000001),A0
        MOV.L #__V_00000001,RP0
```

```
        MOV A0,(RP0+)
        ; --- Restores registers. ---
        POP RP0
        POP A0
        ; --- Performs conditional branches.
        JMP ZERO, #__L_00000011
        PUSH A0
        PUSH RP0
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_00000012
__L_00000011:
        PUSH A0
        PUSH RP0
        MOV.L (#__V_00000001),A0
        MOV.L #__V_00000002,RP0
        MOV A0,(RP0+)
        MOV.L #__V_00000003,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_00000005
__L_00000012:
        JMP #__L_00000005
        JMP #__L_00000005
__L_00000014:
__L_00000005:

SECTION DATA NAME SAREA_DSPDATA
__C_00000000:    DATA H'00000000
__C_00000001:    DATA H'00000001
__C_00000002:    DATA H'00000002
__C_00000003:    DATA H'00000003
__V_00000001:    DATA H'00000000
__V_00000002:    DATA H'00000000
__V_00000003:    DATA H'00000000
__V_00000004:    DATA H'00000000
```

Table 4.11 while Control Statement

Conditional iteration: while … endwh	
Format:	
while∆[expression] statement endwh	
Function:	
The while statements are used to iterate controlling a program while the result of an expression is true. 1) While the evaluation result of an expression is true, the descriptions of statements are iteratively executed.	
Specifications:	
a) Variables, constants, and operators can be described in expressions. b) The following operators are available for expressions. • Relational operators (<, >, <=, >=, <.s, >.s, <=.s, >=.s) • Equality operators (==, !=) • Logical AND/OR operators (&&, ||) • Bitwise AND/OR/exclusive-OR operators (&, |, ^) c) When 'forever' is described in an expression, an infinite-loop code is generated. d) 'endwh' cannot be omitted.	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)

while [A0 < 9]
++A0
endwh
	__L_00000001:
	; --- Saves registers. ---
	PUSH A0
	PUSH RP0
	; --- Acquires operand 2. ---
	PUSH RP0
	PUSH A0
	MOV.L #__V_00000002,RP0
	MOV.L (#__C_00000009),A0
	MOV A0,(RP0+)
	POP A0
	POP RP0
	; --- Acquires operand 1. ---
	PUSH A0
	POP A0
	; --- Executes comparison. ---
	PUSH A0
	PUSH L0
	PUSH L1
	PUSH DP1
	MOV A0,L0
	MOV.L #__V_00000002,DP1
	MOV (DP1+),L1

```
        CLAMP UNSIGNED
        POP DP1
        POP L1
        POP L0
        POP A0
        ; --- Stores the result of comparison. ---
        MOV.L (#__C_00000001),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        ; --- Restores registers. ---
        POP RP0
        POP A0
        ; --- Performs conditional branches. ---
        JMP UNDER, #__L_00000005
        ;--- Stores the result of comparison. ---
        PUSH A0
        PUSH RP0
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        ; --- Performs conditional branches. ---
        JMP #__L_00000003
__L_00000005:
        ; --- Saves registers. ---
        PUSH R0
        PUSH RP0
        ; --- Increments a value. ---
        MOV.L (#__C_00000001),R0
        ADD_R
        ; --- Stores a result. ---
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        ; --- Updates a result. ---
        PUSH DP0
        MOV.L #__V_00000001,DP0
        MOV (DP0+),A0
        PUSH A0
        POP A0
        POP DP0
        ; --- Restores registers. ---
        POP RP0
        POP R0
        PUSH    A0
        PUSH    RP0
```

```
        MOV.L       (#__V_00000001),A0
        MOV.L       #__V_00000002,RP0
        MOV             A0,(RP0+)
        MOV.L       #__V_00000003,RP0
        MOV         A0,(RP0+)
        POP         RP0
        POP         A0
        JMP #__L_00000001
__L_00000006:
__L_00000004:

SECTION DATA NAME SAREA_DSPDATA
__C_00000000:       DATA H'00000000
__C_00000001:       DATA H'00000001
__C_00000009:       DATA H'00000009
__V_00000001:       DATA H'00000000
__V_00000002:       DATA H'00000000
__V_00000003:       DATA H'00000000
__V_00000003:       DATA H'00000000
__V_00000004:       DATA H'00000000
```

Table 4.12 do Control Statement

Conditional iteration (post-execution determination): do … during	
Format:	
do statement during∆[expression]	
Function:	
The do statements are used to iterate controlling a program while the result of an expression is true. The while statements determine whether or not iteration is continued before executing the statements; the do statements determine whether or not iteration is continued after executing the statements. 1) When an expression is evaluated after executing the descriptions of statements and while the evaluation result is true, the descriptions of statements are iteratively executed.	
Specifications:	
a) Variables, constants, and operators can be described in expressions. b) The following operators are available for expressions. • Relational operators (<, >, <=, >=, <.s, >.s, <=.s, >=.s) • Equality operators (==, !=) • Logical AND/OR operators (&&, ||) • Bitwise AND/OR/exclusive-OR operators (&, |, ^) c) When 'forever' is described in an expression, an infinite-loop code is generated.	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)

do
++A0
during [A0 < 9]
	__L_00000001:
	; --- Saves registers. ---
	PUSH R0
	PUSH RP0
	; --- Increments a value. ---
	MOV.L (#__C_00000001),R0
	ADD_R
	; --- Stores a result. ---
	MOV.L #__V_00000001,RP0
	MOV A0,(RP0+)
	; --- Updates a result. ---
	PUSH DP0
	MOV.L #__V_00000001,DP0
	MOV (DP0+),A0
	PUSH A0
	POP A0
	POP DP0
	; --- Restores registers. ---
	POP RP0
	POP R0
	PUSH A0
	PUSH RP0

```
MOV.L (#__V_00000001),A0
MOV.L #__V_00000002,RP0
MOV A0,(RP0+)
MOV.L #__V_00000003,RP0
MOV A0,(RP0+)
POP RP0
POP A0
; --- Saves registers. ---
PUSH A0
PUSH RP0
; --- Acquires operand 2. ---
PUSH RP0
PUSH A0
MOV.L #__V_00000002,RP0
MOV.L (#__C_00000009),A0
MOV A0,(RP0+)
POP A0
POP RP0
; --- Acquires operand 1. ---
PUSH A0
POP A0
; --- Executes comparison. ---
PUSH A0
PUSH L0
PUSH L1
PUSH DP1
MOV A0,L0
MOV.L #__V_00000002,DP1
MOV (DP1+),L1
CLAMP UNSIGNED
POP DP1
POP L1
POP L0
POP A0
; --- Stores the result of comparison. ---
MOV.L (#__C_00000001),A0
MOV.L #__V_00000001,RP0
MOV A0,(RP0+)
; --- Restores registers. ---
POP RP0
POP A0
; --- Performs conditional branches. ---
JMP UNDER, #__L_00000005
;--- Stores the result of comparison. ---
PUSH A0
PUSH RP0
```

```
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        ; --- Performs conditional branches. ---
        JMP #__L_00000006
__L_00000005:
        JMP #__L_00000001
__L_00000006:
__L_00000004:

SECTION DATA NAME SAREA_DSPDATA
__C_00000000:       DATA H'00000000
__C_00000001:       DATA H'00000001
__C_00000009:       DATA H'00000009
__V_00000001:       DATA H'00000000
__V_00000002:       DATA H'00000000
__V_00000003:       DATA H'00000000
```

Table 4.13 for Control Statement

Conditional iteration (with initialization): for … to … step … endfor	
Format: for∆[expression 1]∆to∆[expression 2]∆step∆[expression 3] statement endfor	
Function: The for statements are used to iterate controlling a program while the evaluation result of expression 2 is true. The for statements can specify the initial value in expression 1 and processing after iteration in expression 3 in addition to continuation conditions of expression 2. Therefore, the for statements are often used when execution of a program is iterated fixed times. 1) While the evaluation result of expression 2 is true, the descriptions of statements are iteratively executed. 2) The for statements can specify the initial value in expression 1 and processing after iteration in expression 3.	
Specifications: a) An expression for assigning the initial value to a variable which controls the for statement is described in expression 1. b) The description of expression 1 can be omitted; in this case, '[]' is used. c) The continuation conditions of the for statements are described in expression 2. d) The description of expression 2 can also be omitted; in this case, '[]' is used. When the description of expression 2 is omitted, expression 1 is executed. However, since continuation conditions are always false, expression 3 and the statements in the for statements will not be executed. e) Variables, constants, and operators can be described in expression 2. f) The following operators are available for expressions. • Relational operators (<, >, <=, >=, <.s, >.s, <=.s, >=.s) • Equality operators (==, !=) • Logical AND/OR operators (&&, ||) • Bitwise AND/OR/exclusive-OR operators (&, |, ^) g) An assignment expression for a variable which controls the for statements is described in expression 3. h) The description of expression 3 can also be omitted; in this case, '[]' is used. i) Labels cannot be used in expressions 1 and 3. j) If a label is described in expression 2, it is assumed as an address of the label. k) 'endfor' cannot be omitted.	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)
for [A0 = 0] to [A0 < 9] step [++A0] ... endfor	; Expression 1 [A0 = 0] MOV.L (#__C_00000000),A0 PUSH RP0 MOV.L #__V_00000001,RP0 MOV A0,(RP0+) POP RP0 PUSH A0 PUSH RP0 MOV.L (#__V_00000001),A0 MOV.L #__V_00000002,RP0 MOV A0,(RP0+)

```
        MOV.L       #__V_00000003,RP0
        MOV         A0,(RP0+)
        POP         RP0
        POP         A0
        JMP #__L_00000002
__L_00000001:       ; Expression 3 [++A0]
        ; --- Saves registers. ---
        PUSH R0
        PUSH RP0
        ; --- Increments a value. ---
        MOV.L (#__C_00000001),R0
        ADD_R
        ; --- Stores a result. ---
        MOV.L #__V_00000004,RP0
        MOV A0,(RP0+)
        ; --- Updates a result. ---
        PUSH DP0
        MOV.L #__V_00000004,DP0
        MOV (DP0+),A0
        PUSH A0
        POP A0
        POP DP0
        ; --- Restores registers. ---
        POP RP0
        POP R0
        PUSH        A0
        PUSH        RP0
        MOV.L         (#__V_00000004),A0
        MOV.L       #__V_00000005,RP0
        MOV             A0,(RP0+)
        MOV.L       #__V_00000006,RP0
        MOV         A0,(RP0+)
        POP         RP0
        POP         A0
        JMP #__L_00000002
__L_00000002:       ; Expression 2 [A0 < 9]
        ; --- Saves registers. ---
        PUSH A0
        PUSH RP0
        ; --- Acquires operand 2. ---
        PUSH RP0
        PUSH A0
        MOV.L #__V_00000008,RP0
        MOV.L (#__C_00000009),A0
        MOV A0,(RP0+)
        POP A0
```

```
        POP RP0
        ; --- Acquires operand 1. ---
        PUSH A0
        POP A0
        ; --- Executes comparison. ---
        PUSH A0
        PUSH L0
        PUSH L1
        PUSH DP1
        MOV A0,L0
        MOV.L #__V_00000008,DP1
        MOV (DP1+),L1
        CLAMP UNSIGNED
        POP DP1
        POP L1
        POP L0
        POP A0
        MOV.L (#__C_00000001),A0
        MOV.L #__V_00000007,RP0
        MOV A0,(RP0+)
        ; --- Restores registers. ---
        POP RP0
        POP A0
        ; --- Performs conditional branches. ---
        JMP UNDER, #__L_00000006
        PUSH A0
        PUSH RP0
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000007,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_00000007
__L_00000006:
        JMP #__L_00000001
__L_00000007:
__L_00000005:

SECTION DATA NAME SAREA_DSPDATA
__C_00000000:       DATA H'00000000
__C_00000001:       DATA H'00000001
__C_00000009:       DATA H'00000009
__V_00000001:       DATA H'00000000
__V_00000002:       DATA H'00000000
__V_00000003:       DATA H'00000000
__V_00000004:       DATA H'00000000
```

	__V_00000005: DATA H'00000000 __V_00000006: DATA H'00000000 __V_00000007: DATA H'00000000 __V_00000008: DATA H'00000000

Table 4.14 goto Control Statement

Unconditional branches: goto	
Format:	
goto∆label	
Function:	
The goto statements are used to resume the control of a program at the specified label. 1) The control of a program is resumed at the specified label.	
Specifications:	
a) If a label that has been specified as the destination target of goto does not exist, an assembler error will occur.	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)
goto _L1 ... _L1:	JMP #_L1 ... _L1:

Table 4.15 continue Control Statement

Continuation of iteration processing: continue	
Format:	
continue	
Function:	
The continue statements are used to restart the control of a program from iteration of the innermost loop (while, do … during, or for) including lines describing continue. 1) The control of a program is restarted from iteration of the innermost loop including lines describing continue.	
Specifications:	
a) The control of a program is restarted from iteration of the innermost loop including lines describing continue even if the iteration loop consists of the nest. b) If continue is described at the outside of iteration, an assembler error will occur.	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)
while [forever] ...(1)... if [A0 < 9] continue endif ...(2)... endwh	__L_00000001:; while [forever] PUSH A0 PUSH RP0 PUSH RP0 PUSH A0 MOV.L #__V_00000002,RP0 MOV.L (#__C_00000001),A0 MOV A0,(RP0+) POP A0 POP RP0 MOV.L (#__C_00000000),A0 PUSH DP0 MOV.L #__V_00000002,DP0 CMP (DP0+) POP DP0 MOV.L (#__C_00000001),A0 MOV.L #__V_00000001,RP0 MOV A0,(RP0+) POP RP0 POP A0 JMP NOT_ZERO, #__L_00000005 PUSH A0 PUSH RP0 MOV.L (#__C_00000000),A0 MOV.L #__V_00000001,RP0 MOV A0,(RP0+) POP RP0 POP A0 JMP #__L_00000003 __L_00000005:

```
...(1)...
; --- Saves registers. ---
PUSH A0
PUSH RP0
; --- Acquires operand 2. ---
PUSH RP0
PUSH A0
MOV.L #__V_00000002,RP0
MOV.L (#__C_00000009),A0
MOV A0,(RP0+)
POP A0
POP RP0
; --- Acquires operand 1. ---
PUSH A0
POP A0
; --- Executes comparison. ---
PUSH A0
PUSH L0
PUSH L1
PUSH DP1
MOV A0,L0
MOV.L #__V_00000002,DP1
MOV (DP1+),L1
CLAMP UNSIGNED
POP DP1
POP L1
POP L0
POP A0
; --- Stores the result of comparison. ---
MOV.L (#__C_00000001),A0
MOV.L #__V_00000001,RP0
MOV A0,(RP0+)
; --- Restores registers. ---
POP RP0
POP A0
; --- Performs conditional branches.
JMP UNDER, #__L_00000009
; --- Stores the result of comparison. ---
PUSH A0
PUSH RP0
MOV.L (#__C_00000000),A0
MOV.L #__V_00000001,RP0
MOV A0,(RP0+)
POP RP0
POP A0
; --- Performs conditional branches. ---
```

```
        JMP #__L_0000000a
__L_00000009:
        JMP #__L_00000001
        JMP #__L_00000008
__L_0000000a:
__L_00000008:
        ...(2)...
        JMP #__L_00000001
__L_00000003:
__L_00000004:       ; endwh

SECTION DATA NAME SAREA_DSPDATA
__C_00000000:       DATA H'00000000
__C_00000001:       DATA H'00000001
__C_00000009:       DATA H'00000009
__V_00000001:       DATA H'00000000
__V_00000002:       DATA H'00000000
```

Table 4.16 break Control Statement

Exit of iteration processing and switch statements: break	
Format:	
break	
Function:	
The break statements have the following two functions. 1) Iteration of the innermost loop (while, do … during, or for) including lines describing break is suspended. 2) Execution of the switch … case statements are exited and the control of a program is resumed at the next line of 'endsw'.	
Specifications:	
a) Iteration of the innermost loop including lines describing break is suspended even if the iteration loop consists of the nest. b) If break is described at the outside of iteration or the switch statements, an assembler error will occur. c) Even if the switch statements are described in iteration, break described in the switch statements functions to exit the switch statements. d) Even if iteration is described in the switch statements, break described in iteration functions to exit iteration.	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)
while [forever] ...(1)... if [A0 < 9] break endif ...(2)... endwh	; while __L_00000001: PUSH A0 PUSH RP0 PUSH RP0 PUSH A0 ; [forever] MOV.L #__V_00000002,RP0 MOV.L (#__C_00000001),A0 MOV A0,(RP0+) POP A0 POP RP0 MOV.L (#__C_00000000),A0 PUSH DP0 MOV.L #__V_00000002,DP0 CMP (DP0+) POP DP0 MOV.L (#__C_00000001),A0 MOV.L #__V_00000001,RP0 MOV A0,(RP0+) POP RP0 POP A0 JMP NOT_ZERO, #__L_00000005 PUSH A0 PUSH RP0 MOV.L (#__C_00000000),A0

```
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_00000003
__L_00000005:
        ...(1)...
        ; if [ A0 < 9 ]
        ; --- Saves registers. ---
        PUSH A0
        PUSH RP0
        ; --- Acquires operand 2. ---
        PUSH RP0
        PUSH A0
        MOV.L #__V_00000002,RP0
        MOV.L (#__C_00000009),A0
        MOV A0,(RP0+)
        POP A0
        POP RP0
        ; --- Acquires operand 1. ---
        PUSH A0
        POP A0
        ;--- Executes comparison. ---
        PUSH A0
        PUSH L0
        PUSH L1
        PUSH DP1
        MOV A0,L0
        MOV.L #__V_00000002,DP1
        MOV (DP1+),L1
        CLAMP UNSIGNED
        POP DP1
        POP L1
        POP L0
        POP A0
        MOV.L (#__C_00000001),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        ; --- Restores registers. ---
        POP RP0
        POP A0
        JMP UNDER, #__L_00000009
        PUSH A0
        PUSH RP0
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000001,RP0
```

```
        MOV A0,(RP0+)
        POP RP0
        POP A0
        JMP #__L_0000000a
__L_00000009:
        JMP #__L_00000004                ; break
        JMP #__L_00000008
__L_0000000a:
__L_00000008:
        ...(2)...
        JMP #__L_00000001
__L_00000003:
__L_00000004:                    ; endwh

SECTION DATA NAME SAREA_DSPDATA
__C_00000000:    DATA H'00000000
__C_00000001:    DATA H'00000001
__C_00000009:    DATA H'00000009
__V_00000001:    DATA H'00000000
__V_00000002:    DATA H'00000000
```

4.5 Bit-Manipulating Instructions

The following bit-manipulating instructions are available for the structured description.

- Setting bits: bset
- Clearing bits: bclr
- Testing bits: btst

Table 4.17 bset Instruction

Setting bits: bset
Format:
1) bset∆bpos, dest 2) bset∆A0_*n*
Function:
The bset instruction is used to set registers and the specific bit of a variable specified with parameters to 1. 1) For bpos, either of the immediate indicating the bit position, the register storing the bit position, or the variable is specified. 2) For dest, registers and variables for setting bits are specified. 3) For A0_*n*, any A0 register bit variable is specified (A0_0 to A0_31).
Specifications:
a) The value specified with bpos must be within the range from 0 to 31. If the value of bpos is outside the range, bits of dest are not changed. b) If bpos is an immediate and the value is outside the range, an assembler error will occur. c) The value of n specified for A0_n must be within the range from 0 to 31. If the value of n is outside the range, an assembler error will occur.

Table 4.18 bclr Instruction

Clearing bits: bclr
Format:
1) bclr∆bpos, dest 2) bclr∆A0_*n*
Function:
The bclr instruction is used to clear registers and the specific bit of a variable specified with parameters to 0. 1) For bpos, either of the immediate indicating the bit position, the register storing the bit position, or the variable is specified. 2) For dest, registers and variables for clearing bits are specified. 3) For A0_*n*, any A0 register bit variable is specified (A0_0 to A0_31).
Specifications:
a) The value specified with bpos must be within the range from 0 to 31. If the value of bpos is outside the range, bits of dest are not changed. b) If the value of bpos is an immediate and the value is outside the range, an assembler error will occur. c) The value of n specified for A0_n must be within the range from 0 to 31. If the value of n is outside the range, an assembler error will occur.

Table 4.19 btst Instruction

Testing bits: btst
Format:
1) btstΔbpos, dest 2) btstΔA0_n
Function:
The btst instruction is used to check registers and the specific bit of a variable specified with parameters and update flags Z and O depending on that value. 1) For bpos, either of the immediate indicating the bit position, the register storing the bit position, or the variable is specified. 2) For dest, registers and variables for testing bits are specified. 3) For A0_*n*, any A0 register bit variable is specified (A0_0 to A0_31). 4) Flags Z and O are changed as follows depending on the value of the bit specified with parameters. Flag Z: The flag becomes 1 and 0 when the specified bit is 0 and other than 0, respectively. Flag O: The flag becomes 1 and 0 when the specified bit is 1 and other than 0, respectively.
Specifications:
a) The value specified with bpos must be within the range from 0 to 31. If the value of bpos is outside the range: - when the value of bpos is an immediate, an assembler error will occur. - when the value of bpos is a register or a variable, the value of dest cannot be guaranteed. b) The value of n specified for A0_n must be within the range from 0 to 31. If the value of n is outside the range, an assembler error will occur.

4.6 Concatenating Expressions Using Logical Operators

For the structured description, multiple expressions can be concatenated by using logical operators ('&&' or '||').

Table 4.20 Concatenating Expressions

Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)

if [A0 < M0 && A0 < R0]
...
endif
	; --- Saves registers. ---
	PUSH A0
	PUSH RP0
	;--- Acquires operand 2. ---
	PUSH RP0
	PUSH A0
	PUSH M0
	POP A0
	MOV.L #__V_00000004,RP0
	MOV A0,(RP0+)
	POP A0
	POP RP0
	;--- Acquires operand 1. ---
	PUSH A0
	POP A0
	; --- Executes comparison. ---
	PUSH A0
	PUSH L0
	PUSH L1
	PUSH DP1
	MOV A0,L0
	MOV.L #__V_00000004,DP1
	MOV (DP1+),L1
	CLAMP UNSIGNED
	POP DP1
	POP L1
	POP L0
	POP A0
	; --- Stores the result of comparison. ---
	MOV.L (#__C_00000001),A0
	MOV.L #__V_00000001,RP0
	MOV A0,(RP0+)
	; --- Restores registers. ---
	POP RP0
	POP A0
	; --- Performs conditional branches. ---
	JMP UNDER, #__L_00000004 ; true side
	; --- Stores the result of comparison. ---

```
        PUSH A0
        PUSH RP0
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000001,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        ; --- Performs conditional branches. ---
        JMP #__L_00000007               ; false side
__L_00000004:
        ; --- Saves registers. ---
        PUSH A0
        PUSH RP0
        ;--- Acquires operand 2. ---
        PUSH RP0
        PUSH A0
        PUSH R0
        POP A0
        MOV.L #__V_00000005,RP0
        MOV A0,(RP0+)
        POP A0
        POP RP0
        ;--- Acquires operand 1. ---
        PUSH A0
        POP A0
        ; --- Executes comparison. ---
        PUSH A0
        PUSH L0
        PUSH L1
        PUSH DP1
        MOV A0,L0
        MOV.L #__V_00000005,DP1
        MOV (DP1+),L1
        CLAMP UNSIGNED
        POP DP1
        POP L1
        POP L0
        POP A0
        ; --- Stores the result of comparison. ---
        MOV.L (#__C_00000001),A0
        MOV.L #__V_00000002,RP0
        MOV A0,(RP0+)
        ; --- Restores registers. ---
        POP RP0
        POP A0
        ; --- Performs conditional branches. ---
```

```
        JMP UNDER, #__L_00000006        ; true side
        ; --- Stores the result of comparison. ---
        PUSH A0
        PUSH RP0
        MOV.L (#__C_00000000),A0
        MOV.L #__V_00000002,RP0
        MOV A0,(RP0+)
        POP RP0
        POP A0
        ; --- Performs conditional branches. ---
        JMP #__L_00000007               ; false side
__L_00000006:
        ; ...
        JMP #__L_00000003
__L_00000007:
__L_00000003:

SECTION DATA NAME SAREA_DSPDATA
__C_00000000:       DATA H'00000000
__C_00000001:       DATA H'00000001
__V_00000001:       DATA H'00000000
__V_00000002:       DATA H'00000000
__V_00000003:       DATA H'00000000
__V_00000004:       DATA H'00000000
__V_00000005:       DATA H'00000000
```

4.7 Automatic Generation of Constant Labels

For the structured description, the following three types of data and labels are generated when assembly codes are generated.

- Constant data and constant-data referencing label
- Working variable and working-variable referencing label
- Destination-target referencing label

The constant data and working variables are added in the data section "SAREA_DSPDATA". If the "SAREA_DSPDATA" section does not exist, it is generated automatically. However, a data section allocated in the relocatable form has been defined in the source code, the constant data and working variables are added at the end of the section.

For details of relocatable allocation of the sections, see section 5.4, About Sections.

Table 4.21 Constant Data and Referencing Label

<table>
<tr><td colspan="2">Constant data and referencing label</td></tr>
<tr><td colspan="2">Usage:</td></tr>
<tr><td colspan="2">When constants are used for the structured description, e.g., comparison of integer values in the if statements and initialization of variables in the for statements, constant data and referencing labels are generated to refer to those values.</td></tr>
<tr><td colspan="2">Specifications:</td></tr>
<tr><td colspan="2">a) The naming rule of a label name is defined as follows.
A character string representing "'__C_' + constant value" as eight-digit hexadecimal
b) Even if a constant is represented as decimal in the structured description, constant-data referencing labels generate a label name in hexadecimal notation.</td></tr>
<tr><td colspan="2">Example of expansion of structured descriptions:</td></tr>
<tr><td>Structured description</td><td>Example of expansion of assemblers
(Since this example is a current sample, it may be changed later.)</td></tr>
<tr><td>A0 = 10</td><td>MOV.L (#__C_0000000a),A0 ;A0 = 10
;;; Partially omitted. ;;;

SECTION DATA NAME SAREA_DSPDATA
__C_0000000a: DATA H'0000000a ; Constant label of 10
;;; The rest omitted. ;;;</td></tr>
</table>

Table 4.22 Working Variable and Referencing Label

Working variable and referencing label	
Usage:	
When a working area is required for the structured description, working variable data and referencing labels are generated to allocate the value as a variable.	
Specifications:	
a) The naming rule of a label name is defined as follows. A character string representing "'__V_' + internal ID value" as eight-digit hexadecimal	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)

if [A0 < R0]
R0 = A0
else
A0 = R0
endif
	; if [A0 < R0]
	;--- Saves registers. ---
	PUSH A0
	PUSH RP0
	;--- Acquires operand 2. ---
	PUSH RP0
	PUSH A0
	PUSH R0
	POP A0
	MOV.L #__V_00000002,RP0 ; Working variable which stores the value of operand 2
	MOV A0,(RP0+)
	POP A0
	POP RP0
	;--- Acquires operand 1. ---
	PUSH A0
	POP A0
	;--- Executes comparison. ---
	PUSH A0
	PUSH L0
	PUSH L1
	PUSH DP1
	MOV A0,L0
	MOV.L #__V_00000002,DP1
	MOV (DP1+),L1
	CLAMP UNSIGNED
	POP DP1
	POP L1
	POP L0
	POP A0
	;;; Partially omitted. ;;;
	
	SECTION DATA NAME SAREA_DSPDATA
	__C_00000000: DATA H'00000000
	__C_00000001: DATA H'00000001

	__V_00000001: DATA H'00000000 ; Label of a working variable __V_00000002: DATA H'00000000 ; Label of a working variable __V_00000003: DATA H'00000000 ; Label of a working variable ;;; The rest omitted. ;;;

Table 4.23 Destination-Target Referencing Label

Destination-target referencing label	
Usage:	
To implement control structures such as conditional branches and iteration, labels indicating the destination target of the jmp directive are automatically generated for the structured description.	
Specifications:	
a) The naming rule of a label name is defined as follows. A character string representing "'__L_' + internal ID value" as eight-digit hexadecimal	
Example of expansion of structured descriptions:	
Structured description	Example of expansion of assemblers (Since this example is a current sample, it may be changed later.)
while [A0 < 9] ++A0 endwh	__L_00000001: ; Start label of while [A0 < 9] ;;; Partially omitted. ;;; JMP #__L_00000001 ;endwh __L_00000003: __L_00000004: ; End label of while [A0 < 9] SECTION DATA NAME SAREA_DSPDATA ;;; The rest omitted. ;;;

4.8 Stack Areas Used for the Structured Description

When the structured description is used, the user must allocate suitable stack areas and set the address in the stack area to the SP0 register.

The stack size required for the structured description is described below. When a stack is used in the user application, the user must allocate the size that the following size has been added.

- When codes are generated for GREEN DSP Ver. 2:
  (the number of registers: 9 × 4 bytes) × 2 sets = 72 bytes
- When codes are generated for GREEN DSP Ver. 3:
  (the number of registers: 10 × 4 bytes) × 2 sets = 80 bytes

Table 4.24 Example of the Description for Setting a Stack

Example of Assembler Description:

```
SECTION CODE LOCATE H'mmm
        MOV #STACK_TOP,SP0 ; Initializes a stack pointer.
        …

 ; Allocates a stack area (80 bytes for GREEN DSP Ver. 3)
 SECTION DATA LOCATE H'nnn
 STACK_END:   DATA H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
                DATA   H'00000000
 STACK_TOP:   DATA H'00000000
```

4.9 Outputting the Structured Description to a List File

Both the contents of the structured description and the assembly code generated by the structured description are output to a list file. The contents of the structured description are output with triple semicolons, and the generated assembly code is output from the following line.

```
;
; Description of a source code
MOV   #val, DP0
A0 = (DP0)
MOV (DP0+),A0
```

```
;
; Output to a list file
MOV   #val, DP0

;;; A0 = (DP0)
PUSH DP0
MOV (DP0+),A0
POP DP0
PUSH RP0
MOV.L #__V_00000001,RP0
MOV A0,(RP0+)
POP RP0
PUSH A0
PUSH RP0
MOV.L (#__V_00000001),A0
MOV.L #__V_00000002,RP0
MOV A0,(RP0+)
MOV.L #__V_00000003,RP0
MOV A0,(RP0+)
POP RP0
```

Figure 4.1 Example of Outputting the Structured Description to a List File

5. Overview of Assembling

For assembling in the DSPASM, assembly-source codes which have been described by the user and generated by the structured description are converted to instruction codes for GREEN DSP and the result is output to a file.

This chapter describes the overview of assembling.

For details on assembling, refer to chapter 8, Details of Assembling.

5.1 Specifications of Conversion of Assembly Codes

In the DSPASM, instructions described in the assembly-source code are converted to instruction codes for GREEN DSP.

Values of the available assembly codes and the instruction codes to be converted differ depending on the core version of DSP.

Correspondence between the core version of DSP and the available assembly codes is shown below.

Table 5.1 Assembly and Instruction Codes when the Green-DSP V2 Core is Specified

GREEN-DSP V2 Core: When "2" is specified for the argument of command-line option "-core_version"				
Type	**Instruction**	**Operand 1**	**Operand 2**	**Instruction Code (Hexadecimal)**
Transfer	MOV	A0	M0	1
	MOV	A0	M1	2
	MOV	A0	R0	3
	MOV	A0	L0	5
	MOV	A0	L1	6
	MOV	#XX	DP0	80+XX
	MOV	#XX	DP1	84+XX
	MOV	#XX	RP0	88+XX
	MOV	(DP0+)	A0	7
	MOV	(DP0-)	A0	21
	MOV	(DP1+)	M0	8
	MOV	(DP1+)	M1	9
	MOV	(DP1+)	R0	0a
	MOV	(DP1+)	L0	0b
	MOV	(DP1+)	L1	0c
	MOV	A0	(RP0+)	0d
	MOV	A0	(RP0-)	22
Arithmetic operation	MUL			0e
	ADD			0f
	SUB			11
	MUL_ADD			12
	MUL_SUB			14
	LIMIT			1c
Comparison	CMP	(DP0+)		1d
Branch	JMP	OVER	#XX	cXXX
	JMP	UNDER	#XX	dXXX
	JMP	ZERO	#XX	eXXX
	JMP	NOT_ZERO	#XX	fXXX
	JMP	#XX		bXXX
I/O	OUT	A0	(H'XX)	1eXX
	IN	(H'XX)	A0	1fXX
Control	NOP			0
	STOP			20
Stack manipulation	MOV	#XX	SP0	90+XX
	PUSH	A0		2c
	PUSH	M0		2d
	PUSH	M1		2e
	PUSH	R0		2f
	PUSH	L0		30
	PUSH	L1		31
	PUSH	DP0		32

	PUSH	DP1		33
	PUSH	RP0		34
	POP	A0		35
	POP	M0		36
	POP	M1		37
	POP	R0		38
	POP	L0		39
	POP	L1		3a
	POP	DP0		3b
	POP	DP1		3c
	POP	RP0		3d
Subroutine	JSR	#XX		aXXX
	RET			3e
Logic operation	OR	A0	R0	23
	AND	A0	R0	24
	XOR	A0	R0	25
	NOT	A0		26
	ABS	A0		27
	ABS_S	A0		4f
	SFT_RL			28
	SFT_RA			29
	SFT_LL			2a
	SFT_LA			2b
Extended transfer	MOV	(XX, DP0)	A0	41XX
	MOV	(XX, DP1)	M0	42XX
	MOV	(XX, DP1)	M1	43XX
	MOV	(XX, DP1)	R0	44XX
	MOV	(XX, DP1)	L0	45XX
	MOV	(XX, DP1)	L1	46XX
	MOV	A0	(XX, RP0)	47XX
	MOV	(#XX)	A0	8C+XX
	MOV	(#XX)	M0	94+XX
	MOV	(#XX)	M1	98+XX
	MOV	(#XX)	R0	9C+XX
	MOV	SP0	RP0	48
	MOV	RP0	SP0	49
Non-saturation arithmetic operation	MUL_R			4a
	ADD_R			4b
	SUB_R			4c
	MUL_ADD_R			4d
	MUL_SUB_R			4e
Interrupt	CLI			51
	STI			52
	RETI			54
Others	CODE	H'XX		XX (value of operand)

	JIV			53

Note: In the table, "constant+XX" means an instruction code that a constant value has been added to XX. In addition, "constant value XX" means that the multi-byte instruction codes are generated in combination with the constant value and XX.

Table 5.2 Assembly and Instruction Codes when the Green-DSP V3 Core is Specified

GREEN-DSP V3 Core: When "3" is specified for the argument of command-line option "-core_version"				
Type	**Instruction**	**Operand 1**	**Operand 2**	**Instruction Code (Hexadecimal)**
Transfer	MOV	A0	M0	1
	MOV	A0	M1	2
	MOV	A0	R0	3
	MOV	A0	R1	4
	MOV	A0	L0	5
	MOV	A0	L1	6
	MOV	#XX	DP0	80+XX
	MOV	#XX	DP1	84+XX
	MOV	#XX	RP0	88+XX
	MOV.S	#XX	DP0	80+XX
	MOV.S	#XX	DP1	84+XX
	MOV.S	#XX	RP0	88+XX
	MOV.L	#XX	DP0	[68-6b]80+XX
	MOV.L	#XX	DP1	[68-6b]84+XX
	MOV.L	#XX	RP0	[68-6b]88+XX
	MOV	(DP0+)	A0	7
	MOV	(DP0-)	A0	21
	MOV	(DP1+)	M0	8
	MOV	(DP1+)	M1	9
	MOV	(DP1+)	R0	0a
	MOV	(DP1+)	L0	0b
	MOV	(DP1+)	L1	0c
	MOV	A0	(RP0+)	0d
	MOV	A0	(RP0-)	22
Arithmetic operation	MUL			0e
	ADD			0f
	SUB			11
	MUL_ADD			12
	MUL_ADD3			13
	MUL_SUB			14
	MUL_LIMIT			15
	LIMIT_ADD			16
	LIMIT_ADD3			17
	LIMIT_SUB			18
	MUL_LIMIT_ADD			19
	MUL_LIMIT_ADD3			1a
	MUL_LIMIT_SUB			1b
	MUX			0e
	ADD3			10
	MUX_ADD			12
	MUX_ADD3			13

	MUX_SUB			14
	MUX_CLAMP			15
	CLAMP_ADD			16
	CLAMP_ADD3			17
	CLAMP_SUB			18
	MUX_CLAMP_ADD			19
	MUX_CLAMP_ADD3			1a
	MUX_CLAMP_SUB			1b
	CLAMP			1c
	CLAMP	UNSIGNED		78
	LIMIT			1c
	LIMIT	UNSIGNED		78
Comparison	CMP	(DP0+)		1d
	CMP	UNSIGNED	(DP0+)	79
Branch	JMP	OVER	#XX	[64-67]cXXX
	JMP	UNDER	#XX	[64-67]dXXX
	JMP	ZERO	#XX	[64-67]eXXX
	JMP	NOT_ZERO	#XX	[64-67]fXXX
	JMP	#XX		[64-67]bXXX
I/O	OUT	A0	(H'XX)	1eXX
	IN	(H'XX)	A0	1fXX
Control	NOP			0
	STOP			20
Stack manipulation	MOV	#XX	SP0	90+XX
	MOV.S	#XX	SP0	90+XX
	MOV.L	#XX	SP0	[6c-6f]90+XX
	PUSH	A0		2c
	PUSH	M0		2d
	PUSH	M1		2e
	PUSH	R0		2f
	PUSH	R1		3f
	PUSH	L0		30
	PUSH	L1		31
	PUSH	DP0		32
	PUSH	DP1		33
	PUSH	RP0		34
	POP	A0		35
	POP	M0		36
	POP	M1		37
	POP	R0		38
	POP	R1		40
	POP	L0		39
	POP	L1		3a
	POP	DP0		3b
	POP	DP1		3c

	POP	RP0		3d
Subroutine	JSR	#XX		[64-67]aXXX
	RET			3e
Logic operation	OR	A0	R0	23
	AND	A0	R0	24
	XOR	A0	R0	25
	NOT	A0		26
	ABS	A0		27
	ABS_S	A0		4f
	SFT_RL			28
	SFT_RA			29
	SFT_LL			2a
	SFT_LA			2b
Extended comparison	CMPX	(DP0+)		50
	CMPX	UNSIGNED	(DP0+)	7a
Extended transfer	MOV	(XX, DP0)	A0	41XX
	MOV	(XX, DP1)	M0	42XX
	MOV	(XX, DP1)	M1	43XX
	MOV	(XX, DP1)	R0	44XX
	MOV	(XX, DP1)	L0	45XX
	MOV	(XX, DP1)	L1	46XX
	MOV	A0	(XX, RP0)	47XX
	MOV	(#XX)	A0	8C+XX
	MOV	(#XX)	M0	94+XX
	MOV	(#XX)	M1	98+XX
	MOV	(#XX)	R0	9C+XX
	MOV.S	(#XX)	A0	8C+XX
	MOV.S	(#XX)	M0	94+XX
	MOV.S	(#XX)	M1	98+XX
	MOV.S	(#XX)	R0	9C+XX
	MOV.L	(#XX)	A0	[68-6b]8C+XX
	MOV.L	(#XX)	M0	[68-6b]94+XX
	MOV.L	(#XX)	M1	[68-6b]98+XX
	MOV.L	(#XX)	R0	[68-6b]9C+XX
	MOV	SP0	RP0	48
	MOV	RP0	SP0	49
Non-saturation arithmetic operation	MUX_R			4a
	MUL_R			4a
	ADD_R			4b
	SUB_R			4c
	MUX_ADD_R			4d
	MUL_ADD_R			4d
	MUX_SUB_R			4e
	MUL_SUB_R			4e
Interrupt	CLI			51

	STI			52
	RETI			54
Extended operation	INC	A0		55
	DEC	A0		56
	DIV			58
Table reference	MOV	(DP0+R0)	A0	57
Extended bit manipulation	SFT_RL	n		70
	SFT_LL	n		71
	SFT_RA	n		76
	SFT_LA	n		77
	BTEST	n		72
Flag manipulation	MOV	A0	F0	59
	MOV	F0	A0	5a
	FLGTEST	n		73
	FLGSET	n		74
	FLGCLR	n		75
Segment manipulation	MOV	#XX	PS0	60+XX
	MOVP	#XX	PS0	64+XX
	MOV	#XX	DS0	68+XX
	MOV	#XX	SS0	6c+XX
Others	CODE	H'XX		XX (value of operand)
	JIV			53

Note: In the table, "[constant 1-constant 2]" means that a code for specifying a segment is output before an instruction code when the target address of the instruction is another segment.

5.2 Comments in Assembly Codes

In the DSPASM, the following descriptions are interpreted as comments.

1) From ";" to the end of the line
2) From "//" to the end of the line
3) Description with one or more lines starting with "/*" and ending with "*/"

```
; This line shows a comment.
nop   ; A comment is described from a semicolon to the end of a line.

// This line also shows a comment.
nop   // A comment is described from double slashes to the end of a line.

/* This comment is
*  described across
*  multiple lines.
*/
```

Figure 5.1 Example of Describing a Comment

Japanese (i.e., two-byte character string) can be used for comments. In such a case, use UTF-8 for the Japanese character code.

5.3 Definitions of Data in Data Section

The following shows the format for stating the definitions of data in a data section.

Format:

[character string for the label name:]ΔDATAΔ[decimal, hexadecimal, or #character string for the label name]Δ [comment]

When referring to a label, add "#" to the start of the character string for the label name.

Specifications:

a) For reference to a label name, a label which is defined in the code or data sections can be used. If the label name is not defined, an assembler error will occur.

Sample code:

```
SECTION CODE LOCATE H'0
SUB_START:
        (omitted)

SECTION DATA LOCATE H'1000
S1_ST:  DATA 50              ; DataA
        DATA H'100           ; DataB
        DATA 00000200h       ; DataC
        DATA #SUB_START      ; Address of SUB_START
        DATA #S1_ST          ; Address of S1_ST
```

5.4 Pseudo-Directive in Assembly Codes

In the DSPASM, the following pseudo-directive can be used.

1) .public
2) .line

Table 5.3 .public Pseudo-Directive

.public pseudo-directive
Format:
1) .publicΔsymbol
Function:
Declares a symbol specified with the .public pseudo-directive so that the symbol can be referenced from other modules.
Specifications:
a) Labels can be specified for symbols.

Sample code

```
; Sample code
;
SECTION CODE
      MOV #S1_ST, DP0
      MOV #S1_OUT, RP0
      ;
      MOV (DP0+), A0              ; DataB->A0
      MOV A0, R0                  ; DataB->R0
      MOV (DP0+), A0              ; DataA->A0
      ADD                         ; DataA + DataB
      ;
      MOV A0, (RP0+)              ; A0->DataC
      STOP
      ;
SECTION DATA
S1_ST:
            .public _DataB
_DataB:
            DATA H'00000000       ; DataB
            .public _DataA
_DataA:
            DATA H'00000000       ; DataA
;
S1_OUT:
            .public _DataC
_DataC:
            DATA H'00000000       ; DataC
```

Table 5.4 .line Pseudo-Directive

.line pseudo-directive
Format:
1) .line∆"file name",line number
Function:
The .line pseudo-directive is used to specify the file name and the line number being processed for the DSPASM. 1) The line number begins with 1.
Specifications:
a) The DSPASM does not check that there actually exists the file specified with the .line pseudo-directive. b) The value written to the line number must be within the range from 1 to 100,000. If the value outside the range is written, an assembler error will occur.
Sample code

```
.LINE   "sample.dsp" , 1      ; The following line is the first line of sample.dsp.
; Sample code
;
SECTION CODE
      MOV #S1_ST, DP0
      MOV #S1_OUT, RP0
      ;
.LINE   "includesample.dsp" , 1      ; The following line is the first line of includesample.dsp.
      MOV (DP0+), A0                 ; DataB->A0
      MOV A0, R0                     ; DataB->R0
      MOV (DP0+), A0                 ; DataA->A0
      ADD                            ; DataA + DataB
      ;
.LINE   "sample.dsp" , 7      ; The following line is the seventh line of sample.dsp.
      MOV A0, (RP0+)
      STOP
      ;
SECTION DATA
S1_ST:   DATA H'00000050      ; DataB
         DATA H'00000100      ; DataA
         DATA H'00000003      ; Param_M0
;
S1_OUT: DATA H'00000000
```

5.5 About Sections

An assembly source consists of two types of sections; code and data sections.

When code and data sections are defined, specify CODE and DATA as the section type in the SECTION statement, respectively.

Description of the CODE section
Format:
1) SECTION∆CODE∆[NAME∆section name]∆[LOCATE∆start address]
Function:
A section for allocating program codes is defined. 1) Alphanumeric characters and underscores can be used for the section name. 2) Decimal and hexadecimal constants can be used for the start address.
Specifications:
a) A section name can be omitted. b) When a section name is omitted, another section name immediately before that section name is allocated. However, if there is no section immediately before it, the section name will be SAREA_DSPCODE. c) The start address can be omitted. d) When the start address is specified, the section is allocated in absolute form. e) When the start address is not specified, the section is allocated in relocatable form. f) When multiple sections with the same name are described, they should be allocated so that the addresses are contiguous. g) When sections with the same name have already been defined, the start address cannot be specified. If specified, an assembler error will occur. h) If names of data and code sections are overlapped, an assembler error will occur. i) If address areas are overlapped in multiple sections, an assembler error will occur.

Description of the DATA section
Format:
1) SECTION∆DATA∆[NAME∆section name]∆[LOCATE∆start address]
Function:
A section for allocating program data is defined. 1) Alphanumeric characters and underscores can be used for the section name. 2) Decimal and hexadecimal constants can be used for the start address.
Specifications:
a) A section name can be omitted. b) When a section name is omitted, another section name immediately before that section name is allocated. However, if there is no section immediately before it, the section name will be SAREA_DSPCODE. c) The start address can be omitted. d) When the start address is specified, the section is allocated in absolute form. e) When the start address is not specified, the section is allocated in relocatable form. f) When multiple sections with the same name are described, they should be allocated so that the addresses are contiguous. g) When sections with the same name have already been defined, the start address cannot be specified. If specified, an assembler error will occur. h) If names of data and code sections are overlapped, an assembler error will occur. i) If address areas are overlapped in multiple sections, an assembler error will occur.

5.5.1 Allocating Sections and the Number of Sections

Sections can be allocated in absolute form or relocatable form.

When a section allocated in relocatable form is defined, other sections than that section cannot be defined.

Multiple sections allocated in absolute form can be defined.

The following shows the combination of code and data sections.

	The number of sections definable for code sections	The number of sections definable for data sections
Relocatable code section Relocatable data section	One section only	One section only
Relocatable code section Absolute data section	One section only	Multiple sections
Absolute code section Relocatable data section	Multiple sections	One section only
Absolute code section Absolute data section	Multiple sections	Multiple sections

```
; Sample code
SECTION CODE NAME DSPCODE1   ; This section is allocated in relocatable form.
    NOP
    STOP

SECTION CODE NAME DSPCODE1   ; The section that has already been defined is only available.
    NOP                      ; Note that LOCATE cannot be specified.
    STOP
;
SECTION DATA NAME DSPDATA1 LOCATE H'00004000   ; This section is allocated
S0_ST:   DATA H'00000050                       ; in absolute form.
S0_OUT: DATA H'00000000

SECTION DATA NAME DSPDATA2 LOCATE H'00005000
S2_ST:   DATA H'00000050
S2_OUT: DATA H'00000000

SECTION DATA NAME DSPDATA1   ; The section that has already been defined is available.
S1_ST:   DATA H'00000050     ; Note that LOCATE cannot be specified.
S1_OUT: DATA H'00000000
```

Figure 5.2 Example of the Description of Sections 1 (Code Section: Relocatable, Data Section: Absolute)

```
; Sample code
SECTION CODE NAME DSPCODE1 LOCATE H'00006000   ; This section is allocated
    NOP                                        ; in absolute form.
    STOP

SECTION CODE NAME DSPCODE2 LOCATE H'00007000
    NOP
    STOP

SECTION CODE NAME DSPCODE1   ; When the section that has already been defined is used,
    NOP                      ; LOCATE cannot be specified.
    STOP
;
SECTION DATA NAME DSPDATA1   ; This section is allocated in relocatable form.
S0_ST:   DATA H'00000050
S0_OUT: DATA H'00000000

SECTION DATA NAME DSPDATA1   ; The section that has already been defined is only available.
S1_ST:   DATA H'00000050     ; Note that LOCATE cannot be specified.
S1_OUT: DATA H'00000000
```

Figure 5.3 Example of the Description of Sections 2 (Code Section: Absolute, Data Section: Relocatable)

5.5.2 Note on Defining Multiple Sections

When multiple code and data sections are described and those are transferred from ROM of the CPU to SRAM of the DSP, allocation of sections must not be changed even if there are gaps between sections (do not change the offset from the base address).

5.6 Direct Description of Instruction Codes

The DSPASM supports the description "CODE H'xx" which directly describes instruction codes. When this instruction is described, the value specified with an argument is output as the assembly code.

6. Details of Preprocessing

This chapter explains the details of preprocessing that are not described in chapter 3, Overview of Preprocessing.

6.1 Operators of Constant Expressions

The following shows operators that are available for constant expressions with conditional inclusion preprocessing directive ({TC}if).

Table 6.1 Operators Available for Constant Expressions of Preprocessing Directives

Type	Operator	Description
Unary operators	+	Indicates positive numbers.
	-	Indicates negative numbers.
	~	Performs bitwise-inversion operation (NOT).
Division and multiplication operators	*	Performs multiplication (unsigned).
	*.s	Performs multiplication (signed, saturation operation)
	/	Performs division (unsigned).
	%	Performs residue operation (unsigned).
Addition and subtraction operators	+	Performs addition (unsigned).
	+.s	Performs addition (signed, saturation operation)
	-	Performs subtraction (unsigned).
	-.s	Performs subtraction (signed, saturation operation)
Bitwise shift operators	<<	Performs logical left-shift operation for the specified bits.
	<<.s	Performs arithmetic left-shift operation for the specified bits.
	>>	Performs logical right-shift operation for the specified bits.
	>>.s	Performs arithmetic right-shift operation for the specified bits.
Relational operators	<	The operation result is true when the left side of the expression is smaller than the right side (unsigned).
	<.s	The operation result is true when the left side of the expression is smaller than the right side (signed).
	>	The operation result is true when the left side of the expression is larger than the right side (unsigned).
	>.s	The operation result is true when the left side of the expression is larger than the right side (signed).
	<=	The operation result is true when the left side of the expression is smaller than or equal to the right side (unsigned).
	<=.s	The operation result is true when the left side of the expression is smaller than or equal to the right side (signed).
	>=	The operation result is true when the left side of the expression is larger than or equal to the right side (unsigned).
	>=.s	The operation result is true when the left side of the expression is larger than or equal to the right side (signed).
Equality operators	==	The operation result is true when the left side of the expression is equal to the right side.
	!=	The operation result is true when the left side of the expression is not equal to the right side.
Bitwise AND/OR/ exclusive-OR operators	&	Performs bitwise AND operation (AND).
	|	Performs bitwise OR operation (OR).
	^	Performs bitwise exclusive-OR operation (XOR).
Logical AND/OR operators	&&	The operation result is true when the values of the left and right sides of the expression are compared to 0 and both of them are not equal. Short-circuit evaluation is performed (if the left side is false, the right side is not evaluated).
	||	The operation result is true when the values of the left and right sides of the expression are compared to 0 and either of them is not equal. Short-circuit evaluation is performed (if the left side is true, the right side is not evaluated).

The priority of operators are the same as that described in section 4.3.1, Priority of Operators.

7. Details of Structured-Descriptive Processing

This chapter explains the details of structured-descriptive processing that are not described in chapter 4, Overview of Structured-Descriptive Processing.

7.1 Writing Address Values

An address value can be written as an immediate to address pointer registers. However, it cannot be written to operation parameter registers such as A0 or M0.

When an address value is specified for the operation parameter register, use a register or a variable that the address value has been assigned.

Example of Assembler Description:

```
MOV   #H'000, DP0        ; An address value can be written as an immediate (hexadecimal).
MOV   #100,DP0           ; An address value can be written as an immediate (decimal).
;
MOV   #H'00000000, A0    ; An address value cannot be written as an immediate to the A0 register.
```

7.2 Restrictions on the Structured Description

7.2.1 Expressions over Multiple Lines

In the DSPASM, since the structured description is analyzed in a line unit, expressions over multiple lines cannot be described.

Similarly, elements to determine the structured description (e.g., in the for statement, from the reserved word "for" to [expression 3] which specifies increments) must also be described in a line.

7.2.2 Operators in Control Statements

In the structured description, operators which return true or false values can only be used in expressions in the control statements. If used in the other statements, an assembler error will occur.

Type of Operators Returning True or False Values	Operator
Relational operators	<, <.s, >, >.s, <=, <=.s, >=, >=.s
Equality operators	==, !=
Logical AND/OR operators	&&, ||

Operators other than logical AND/OR cannot be used with expressions for which the result may be true or false. For example, an assembler error will occur in the case of the following code.

Example of Coding for Continuous Operation Following a True or False Result:

```
if [(A0 < R0) == 0]     ; An assembler error will occur since a test for equivalence is applied to the result of the
                          relational operation "A0 < R0".
        (omitted)
endif
```

7.2.3 Bit Manipulation Instructions

Bit manipulation instructions cannot be used in expressions in the control statements or with other operators.

Bit manipulation instructions	bset, bclr, btst

7.2.4 Variables That Cannot Be Handled by Operators

This section describes variables that cannot be handled by operators. If an unusable variable is described, an assembler error will occur.

Type	Operator	Variables That Cannot Be Handled
Unary operators	+	None
	-	The following variables cannot be used: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O A0 register bit variables: A0_0 to A0_31 Pointer variable: (immediate, RP0)
	~	The following variables cannot be used: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O Pointer variable: (immediate, RP0)
	++	The following variables cannot be used: Register variables: PS0, DS0, SS0 Register variables: I, Z, U, O A0 register bit variables: A0_0 to A0_31 Pointer variables: (DP0), (DP1), (immediate, DP0), (immediate, DP1), (immediate, RP0), (DP0+R0) Constants
	++.s	
	--	
	--.s	
Division and multiplication operators	*	The following variables cannot be used: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O A0 register bit variables: A0_0 to A0_31 Pointer variable: (immediate, RP0)
	*.s	
	/	The following variables cannot be used: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O A0 register bit variables: A0_0 to A0_31 Pointer variable: (immediate, RP0)
	%	
Addition and subtraction operators	+	The following variables cannot be used: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O A0 register bit variables: A0_0 to A0_31
	+.s	
	-	
	-.s	

		Pointer variable: (immediate, RP0)
Bitwise shift operators	<<	The following variables cannot be used: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O A0 register bit variables: A0_0 to A0_31 Pointer variable: (immediate, RP0)
	<<.s	
	>>	
	>>.s	
Relational operators	<	The following variables cannot be used: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O A0 register bit variables: A0_0 to A0_31 Pointer variables: (DP0), (DP1), (immediate, DP0), (immediate, DP1), (immediate, RP0), (DP0+R0)
	<.s	
	>	
	>.s	
	<=	
	<=.s	
	>=	
	>=.s	
Equality operators	==	The following variables cannot be used: Register variables: S0, DS0, SS0 Flag variable: I Pointer variable: (immediate, RP0) Flag variables (Z, U, O) and A0 register bit variables can only be used for comparison with 0 or 1.
	!=	
Bitwise AND/OR/exclusive-OR operators	&	The following variables cannot be used on the right side of expressions: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O A0 register bit variables: A0_0 to A0_31 Pointer variable: (immediate, RP0)
	|	
	^	
Logical AND/OR operators	&&	The following variables cannot be used: Register variables: PS0, DS0, SS0 Flag variable: I Pointer variable: (immediate, RP0)
	||	
Assignment operators	=	The following variables cannot be used on the left side of expressions: Flag variables: Z, U, O Pointer variables: (DP0), (DP1), (immediate, DP0), (immediate,DP1), (DP0+R0) The following variables cannot be used on the right side of expressions: Register variables: PS0, DS0, SS0 Flag variables: I, Z, U, O A0 register bit variables Pointer variable: (immediate, RP0) When the following variables are used on the left side, anything except for
	*=	
	*=.s	
	/=	
	%=	
	+=	
	+=.s	
	-=	
	-=.s	
	<<=	

	<<=.s	constants cannot be used on the right side.
	>>=	Register variables: PS0, DS0, SS0
	>>=.s	When the following variables are used on the left side, anything except for 0 and 1 cannot be used on the right side.
	&=	
	|=	Flag variable: I
	^=	A0 register bit variables: A0_0 to A0_31
	Compound assignment (=..=)	

For other restrictions than above, refer to section 10.2, Translation Limits on the Structured Description.

7.3 Crossing Nests in the Structured Description

Nests cannot be crossed in the structured description. If nests are crossed, an assembler error will occur.

Table 7.1 Example of Crossing of a Nest in the Structured Description

Example of Assembler Description:	
while [A<B]	
if [A==C]	
break	
endwh	;;; An assembler error occurs in this line.
endif	

7.4 Differences of Code Generation Depending on the Core Version of DSP

In the structured description, there are the following differences in the assembly code to be generated depending on the core version of DSP which is specified with the -core_version option.

Core Version 2: V2	Core Version 3: V3
1) MUL/MUL_R is used for multiplication.	1) MUX/MUX_R is used for multiplication.
2) MOV is used for data transfer.	2) MOV.L is used for data transfer.
3) LIMIT is used for relational operation.	3) CLAMP is used for relational operation.
	4) The following directives are only used in the V3 core.
	DIV
	BTEST

7.5 Character Sets Available in the Structured Description

The following characters are available in the structured description.

Table 7.2 Character Sets Available in the Structured Description

Available Character Sets	
Item	**Value**
English uppercase characters	A B C D E F G H I J K L M N O P Q R S V W X Y Z
English lowercase characters	a b c d e f g h i j k l m n o p q r s t u v w x y z
Numerals	0 1 2 3 4 5 6 7 8 9
Special characters	! % & ( ) * + , - . / ; < = > [ ] ^ | ~
Space characters	Space tab
New-line characters	CR LF

Note that English upper-case and lower-case characters are not distinguished in the structured description.

7.6 Differences of Meanings at the Spots where "()" is Used

The meanings of "()" differ as shown below depending on the spots where it is used.

Spot where "()" is used	Description
Control statement	Changes the priority of operation.
Other than control statement	Is handled as a pointer variable. If a variable name which is not described in section 4.1.4, Pointer Variables, is used, an assembler error will occur.

Sample code

```
SECTION CODE
    if [(DATA2) == H'100]     // Since (DATA2) is not handled as a pointer variable, the meaning is the same as
                              // DP0 == H'100 and the result is true.

        (DATA2) = H'100       // Since (DATA2) is handled as a pointer variable,
                              // H'100 is set for DATA2.
    endif
    ;
    for [R0 = (DATA1) * 2] to [R0 < 16] step [R0 += 4] // Since (DATA1) is not handled as a pointer variable,
                                                       // H'0000 is set for R0.
        (DATA1) = R0    // Since (DATA1) is handled as a pointer variable,
                        // Value of R0 is set to the 4-byte area starting from DATA1.
    endfor
    STOP
;
SECTION DATA NAME DATASEC1 LOCATE H'0000
DATA1:
    DATA H'00000004
;
SECTION DATA NAME DATASEC2 LOCATE H'0100
DATA2:
    DATA H'00000200
;
```

7.7 Note on Using the Structured Description when the V3 Core is Used

In the structured description, since the MOV.L directive is used in the assembly code to be generated, the value of the DS0 register may be changed at the spot where the structured description is used. Accordingly, when the user must specify the data segment, check the specifications of the data segment in the software manual of GREEN-DSP and modify the assembly code according to the following steps if required.

(1) Assemble the source code and output a list file.
(2) Find the spot where MOV.L is used in the structured-description section from a list file.
(3) Check that the data segment has not been rewritten as an unintended value by the MOV.L directive.
(4) If the data segment has been rewritten as an unintended value, the MOV.S and MOV directives after exiting the structured-description section will refer to an unintended segment.
(5) Therefore, modify the following MOV.S and MOV directives as the MOV.L directive.

7.8 Note on Using the Structured Description without Side Effect

When the structured description without side effect (e.g., expression without assignment) is used, the instruction code that the value of the operation result is set to a working area is generated.

[Example of structured description without side effect]

- A0 + R0
- 10
- 1 + 2

Table 7.3 Example of Expansion of Assemblers when Constant "10" is Described in the Structured Description

Example of Expansion of Assembers

```
     1                  : .line "C:¥dspasm¥sample_code.dsp" , 1
     2                  : SECTION CODE
     3                  :
     4                  : ;;; 10
     5 0000 2c          :    PUSH A0
     6 0001 34          :    PUSH RP0
     7 0002 688c00      :    MOV.L (#__C_0000000a),A0
     8 0005 688801      :    MOV.L #__V_00000001,RP0
     9 0008 0d          :    MOV A0,(RP0+)
    10 0009 3d          :    POP RP0
    11 000a 35          :    POP A0
    12 000b 2c          :    PUSH A0
    13 000c 34          :    PUSH RP0
    14 000d 688c00      :    MOV.L (#__C_0000000a),A0
    15 0010 688802      :    MOV.L #__V_00000002,RP0
    16 0013 0d          :    MOV A0,(RP0+)
    17 0014 3d          :    POP RP0
    18 0015 35          :    POP A0
    19                  :
    20                  : SECTION DATA NAME SAREA_DSPDATA
    21 0000 0000000a : __C_0000000a:       DATA H'0000000a
    22 0004 00000000 : __V_00000001:       DATA H'00000000
    23 0008 00000000 : __V_00000002:       DATA H'00000000
```

8. Details of Assembling

This chapter explains the details of assembling that are not described in chapter 5, Overview of Assembling.

8.1 Restrictions on Assembling

For restrictions on assembling, refer to section 10.3, Translation Limit on Assembling.

8.2 Character Sets Available in the Assembly Description

The following characters are available in the assembly description. Note that upper-case and lower-case characters are not distinguished in the assembly description.

Table 8.1 Character Sets Available in the Assembly Description

Available Character Sets	
Item	**Value**
English uppercase characters	A B C D E F G H I J K L M N O P Q R S V W X Y Z
English lowercase characters	a b c d e f g h i j k l m n o p q r s t u v w x y z
Numerals	0 1 2 3 4 5 6 7 8 9
Special characters	! " # $ ' * ( ) + , - . / : ; ? _ `
Space characters	Space tab
New-line characters	CR LF

The following shows a list of characters and symbols that can be used in label and section names.

If characters that cannot be recognized in label and section names are used, an assembler error will occur.

Table 8.2 Characters and Symbols Available for Use in Label and Section Names

English uppercase characters	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
English lowercase characters	a b c d e f g h i j k l m n o p q r s t u v w x y z
Numerals	0 1 2 3 4 5 6 7 8 9
Special characters	! $ ' . ? _ `

8.3 Supplementary on Generating Assembly Codes

- JMP/JSR directives

  The destination-target address of the JMP/JSR directives can be specified in the 12-bit width. If the specified address exceeds the 12-bit width, the destination-target address will be the value that the specified value has been masked with 0x0FFF.

  In addition, only when the V3 core is in use, upper four bits (values masked with 0xF000) are regarded as segment numbers.

- OUT/IN directives

  Ports for the OUT/IN directives can be specified in the 8-bit width. If the specified port number exceeds the 8-bit width, the port number will be the value that the specified value has been masked with 0xFF.

- Segment manipulation directives (transfer to the PS0, DS0, or SS0 register)

  Segment values specified with the segment manipulation directive must be within the range from 0 to 3. If the specified value exceeds the range, an assembler error will occur.

- CODE directive
  Output codes for the CODE directive can be specified in the 8-bit width. If the specified output code exceeds the 8-bit width, the output code will be the value that the specified value has been masked with 0xFF.

- Overlapped labels
  If labels are overlapped in the DSPASM, an assembler error will occur. The error will also occur when data and code labels are overlapped.

- Global symbols
  If labels are overlapped in the DSPASM, an assembler error will occur. The error will also occur when data and code labels are overlapped. Labels will be local symbols which cannot be referenced from an object file on the CPU.
  To use a label as a global symbol that can be referenced from an object file on the CPU, use .PUBLIC as described below.
  To reference the global symbol as a variable in a C program on the CPU, use the symbol name according to the C language specifications.

Example of Expansion of Assembers
SECTION DATA .PUBLIC _DATA1 ; Declares the label (_DATA1) as a global symbol. _DATA1: DATA H'00000004

9. Reserved Words

The following shows reserved words used for the DSPASM.

Table 9.1 Reserved Words for Predefined Macros (1)

__VA_ARGS__	__RENESAS_VERSION__
__RENESAS__	__DSPASM__

Table 9.2 Reserved Words for Preprocessors (2)

{TC}define	{TC}ifdef
{TC}if	{TC}ifndef
{TC}else	{TC}elif
{TC}endif	{TC}include

Table 9.3 Reserved Words for Structured Descriptions (3)

if	endsw	for	break
elif	default	to	forever
else	while	step	bset
endif	endwh	endfor	bclr
switch	do	goto	btst
case	during	continue	

Table 9.4 Reserved Words for Assembly Descriptions (4)

section	.line
code	
data	
locate	
name	

Table 9.5 Reserved Words for Assembly Directives (5)

ABS	FLGSET	MUL_ADD_R	MUX_SUB_R
ABS_S	FLGTEST	MUL_LIMIT	NOP
ADD	IN	MUL_LIMIT_ADD	NOT
ADD3	INC	MUL_LIMIT_ADD3	OR
ADD_R	JIV	MUL_LIMIT_SUB	OUT
AND	JMP	MUL_R	POP
BTEST	JSR	MUL_SUB	PUSH
CLAMP	LIMIT	MUL_SUB_R	RET
CLAMP_ADD	LIMIT_ADD	MUX	RETI
CLAMP_ADD3	LIMIT_ADD3	MUX_ADD	SFT_LA
CLAMP_SUB	LIMIT_SUB	MUX_ADD3	SFT_LL
CLI	MOV	MUX_ADD_R	SFT_RA
CMP	MOV.L	MUX_CLAMP	SFT_RL
CMPX	MOV.S	MUX_CLAMP_ADD	STI
CODE	MOVP	MUX_CLAMP_ADD3	STOP
DEC	MUL	MUX_CLAMP_SUB	SUB
DIV	MUL_ADD	MUX_R	SUB_R
FLGCLR	MUL_ADD3	MUX_SUB	XOR

Table 9.6 Reserved Words for Assembly Registers (6)

A0	R1	PS0	I
M0	DP0	DS0	Z
M1	DP1	SS0	U
L0	RP0	BR0	O
L1	F0	PG0	
R0	SP0		

Table 9.7 Reserved Words for A0 Bit Variables (7)

A0_0	A0_8	A0_16	A0_24
A0_1	A0_9	A0_17	A0_25
A0_2	A0_10	A0_18	A0_26
A0_3	A0_11	A0_19	A0_27
A0_4	A0_12	A0_20	A0_28
A0_5	A0_13	A0_21	A0_29
A0_6	A0_14	A0_22	A0_30
A0_7	A0_15	A0_23	A0_31

- Other reserved words

  In addition to the reserved words above, a word including double underscores ("__") is also regarded as the reserved word.

10. Translation Limits

The DSPASM has the following translation limits.

10.1 Translation Limits on Preprocessing

Preprocessing has the following translation limits.

Table 10.1 Translation Limits on Preprocessing

Number of descriptions of macro replacement preprocessing directives	Up to 1,024 * Up to 512 in V1.03.00 or earlier
Character string length of identifiers for macro replacement	Up to 200 characters
Number of parameters for function macros	Up to 31
Number of levels of the nest for conditional inclusion ({TC}if/{TC}ifdef/{TC}ifndef/{TC}elif)	Up to 31 nesting levels * Only the source codes that satisfy conditions are to be counted.
Number of nests for file inclusion ({TC}include)	Up to 64 counts
Length of a line (not including new-line codes)	Up to 2,048 characters

If those limits are exceeded, an assembler error will occur.

Table 10.2 Method for Counting Nesting Levels

```
#define   NUM   1
#if   #NUM == 1              ; The most outside preprocessing directive is not counted as a nesting level.
      #if   #NUM > 0         ; Nesting level 1
            #if   #NUM <= 2  ; Nesting level 2
            #endif
      #endif
#endif
```

In a source code that is excluded from inclusion with the conditional inclusion preprocessing directive, the nesting levels are not counted. In such a source code, even if the number of nesting levels exceeds 31, an assembler error will not occur.

Table 10.3 Examples of Descriptions Where Nest Count of Conditional Inclusion Preprocessing Directive Exceeds Translation Limits

```
#define   NUM     0

#if   #NUM == 0
      #if   #NUM < 1                ; Nesting level 1
      -- omitted --
            #if   #NUM < 31         ; Nesting level 31
                  #if   #NUM < 32   ; Nesting level count exceeds 31. An assembler error occurs on this line.
                        ; Processing on nesting level 32
                  #endif
            #endif
      -- omitted --
      #endif
#endif
```

```
#define   ABC     0

#ifdef   ABCD                   ; Since a macro is not defined for ABCD, nests from the next line are not included.
                                ; Nests following #else are included.
      #ifdef   ABC              ; Nesting level 1
      -- omitted --
            #ifdef   ABC        ; Nesting level 31
                  #ifdef   ABC  ; Although the nesting level count exceeds 31, an assembler error does not occur
                                ; because this is not to be included.
                        ; Processing on nesting level 32
                  #endif
            #endif
      -- omitted --
      #endif
#else
      #ifdef   ABC              ; Nesting level 1
      -- omitted --
            #ifdef   ABC        ; Nesting level 31
                  #ifdef   ABC  ; Nesting level count exceeds 31. An assembler error occurs on this line.
                        ; Processing on nesting level 32
                  #endif
            #endif
      -- omitted --
      #endif
#endif
```

```
#define   NUM     31

#if   #NUM > 0
      #if   #NUM > 1        ; Nesting level 1
      -- omitted --
            #if   #NUM > 31             ; Nesting level 31. Since this constant expression is false, the next #if is not
 included.
                  #if   #NUM > 32     ; Although the nesting level count exceeds 31, an assembler error does not occur
                                        ; because this is not to be included.
                        ; Processing on nesting level 32
                  #endif
            #elif   #NUM == 31
                  #ifdef   NUM          ; Nesting level count exceeds 31. An assembler error occurs on this line.
                       ; Processing on nesting level 32
                  #endif
            #else
                  ; Processing on nesting level 31. This line is not included.
            #endif
      -- omitted --
      #endif
#endif
```

```
#define   ABC     0

#ifndef   ABCD                  ; Since a macro is not defined for ABCD, nests from the next line are included.
                                ; Nests following #else are not included.
      #ifdef   ABC              ; Nesting level 1
      -- omitted --
            #ifdef   ABC          ; Nesting level 31
                  #ifdef   ABCD   ; Nesting level count exceeds 31. An assembler error occurs on this line.
                                  ; Even if the condition is false, it is counted as a nest.
                       ; Processing on nesting level 32
                  #endif
            #endif
      -- omitted --
      #endif
#else
      #ifdef   ABC                ; Nesting level 1
      -- omitted --
            #ifdef   ABC          ; Nesting level 31
                  #ifdef   ABC    ; Although the nesting level count exceeds 31, an assembler error does not occur
                                  ; because this is not to be included.
                       ; Processing on nesting level 32
                  #endif
            #endif
      -- omitted --
      #endif
#endif
```

10.2 Translation Limits on the Structured Description

The structured description has the following translation limits.

Table 10.4 Translation Limits on the Structured Description

Length of a line (not including new-line codes)	Up to 2,048 characters
Number of levels of the nest for control statements	Up to 31 nesting levels *Up to 31 nests are specified regardless of the type of control statements.
Number of operators available for one control statement	Up to 31

If those limits are exceeded, an assembler error will occur.

10.3 Translation Limit on Assembling

Assembling has the following translation limit.

Table 10.5 Translation Limit on Assembling

Length of a line (not including new-line codes)	Up to 2,048 characters

If this limit is exceeded, an assembler error will occur.

11. Error Messages

This chapter describes error messages output from the DSPASM.

11.1 Formats of Error Messages

Messages of the assembler are output in the following formats according to the specifications of CubeSuite+.

(1) When the file name and line number are included

```
file-name (line-number) : message-type component-number message-number : message
```

(2) When the file name and line number aren't included

```
message-type component-number message-number : message
```

Note that message types, component numbers, and message numbers are output as the continued character string.

Message types	One alphabetic character ("E" for an error or "W" for a warning)
Component numbers	Two digits (Fixed to "05".)
Message numbers	Five digits (The first digit is fixed to "5".)

11.2 Error Messages

The following shows errors generated in the DSPASM.

If an error occurs, processing will be terminated. In the case of a warning, on the other hand, processing is resumed after a message has been displayed.

E0553001: command line option. (-format)	
Description	The invalid -format option is specified.
Display information	None
Action by user	Specify any one of OBJ, ASM, or VERILOG.

E0553002: command line option. (-dsp)	
Description	The invalid -dsp option is specified.
Display information	None
Action by user	Specify any one of the following: RX_DSP, RL78_DSP, RL78_101_DSP, RL78_111_DSP, RL78_IAR_DSP, RL78_LLVM_DSP, RL78_GCC_DSP, ARM_DSP, or ARM_EABI5_DSP

E0553003: command line option. (-core_version)	
Description	The invalid -core_version option is specified.
Display information	None
Action by user	Specify either 2 or 3.

E0553004: command line option. (-text_macro)	
Description	An unusable character is specified for the -text_macro option.
Display information	None
Action by user	Specify any of the following characters: # ' ` @ _

E0553005: command line option. (*specified option name*)	
Description	A non-existent option is specified.
Display information	The specified option name
Action by user	Check specification of options.

E0553006: illegal file name.	
Description	The name of the assembly-language file was illegal (a wrong extension, etc.).
Display information	The name of the assembly-language file which has been specified for input
Action by user	Check the name of the assembly-language file.

E0553007: illegal include file name.	
Description	The name of the file to be included was illegal ('>' was not found, no file name was described, etc.).
Display information	The name of the file to be included and the line number where an error occurred
Action by user	Check the name of the file to be included.

E0553008: source file open error.	
Description	No assembly-language file could not be read.
Display information	The name of the assembly-language file which has been specified for input and the line number where an error occurred
Action by user	Check that the assembly-language file can be read.

E0553009: include file open error.	
Description	No file to be included could not be read.
Display information	The name of the file to be included and the line number where an error occurred
Action by user	Check that the file to be included can be read.

E0553010: include file nesting over.	
Description	The number of nests for the include file exceeds the upper limit (64).
Display information	The name of the file to be included and the line number where an error occurred
Action by user	Check that the nest structure of the include file is not circulated.

E0553011: file write error.	
Description	Writing to the output file was failed.
Display information	The name of the output file
Action by user	Check that the output file can be written.

E0553012 : define symbol not found.	
Description	No identifier is specified in the description of {TC}define.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that identifiers have been defined.

E0553013: text macro redefined.	
Description	A text macro was redefined. *If the command-line option "-allow_ text_macro_redefine" is specified, this error will not occur.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that definitions of identifiers are not overlapped.

E0553014: illegal ifdef.	
Description	No identifier is specified in the description of {TC}ifdef.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the line where an error occurred.

E0553015: illegal ifndef.	
Description	No identifier is specified in the description of {TC}ifndef.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that identifiers have been defined.

E0553016: illegal if.	
Description	No identifier is specified in the description of {TC}if.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that constant statements have been described.

E0553017: illegal elif.	
Description	No identifier is specified in the description of {TC}elif.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that constant statements have been described.

E0553018: illegal else.	
Description	No preprocessing directive for {TC}else was found.
Display information	The name of the file and the line number where an error occurred
Action by user	Describe any one of {TC}if, ifdef, or ifndef before {TC}else.

E0553019: illegal endif.	
Description	No preprocessing directive for {TC}endif was found.
Display information	The name of the file and the line number where an error occurred
Action by user	Describe any one of {TC}if, ifdef, or ifndef before {TC}endif.

E0553020: if - endif not found.	
Description	No {TC}endif for {TC}if was found.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that {TC}endif for {TC}if has been described.

E0553021: ifdef - endif not found.	
Description	No {TC}endif for {TC}ifdef was found.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that {TC}endif for {TC}ifdef has been described.

E0553022: ifndef - endif not found.	
Description	No {TC}endif for {TC}ifndef was found.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that {TC}endif for {TC}ifndef has been described.

E0553023: cannot allocate memory.	
Description	Allocation of dynamic memory was failed.
Display information	The name of the file and the line number where an error occurred
Action by user	When other programs are running on Windows, exit them and reexecute the DSPASM. In addition, check the description of the line where an error occurred.

E0553024: illegal define. (*identifier name*)	
Description	There was an illegal description in {TC}define.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the line where an error occurred.

E0553025: illegal expression.	
Description	There was an illegal description in the constant expression.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the constant expression.

E0553026: constant value overflow.	
Description	A constant value exceeds the upper limit (4 bytes).
Display information	The name of the file and the line number where an error occurred
Action by user	Specify a value from 0 to 4294967295 for a constant.

E0553027: zero divide.	
Description	Division by 0 occurred when the description of the constant expression was processed.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the constant expression.

E0553028: unexpected EOF.	
Description	An unexpected EOF was detected.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the following for the description of the line where an error occurred. • Comments over multiple lines are closed.

E0553029: unknown section.	
Description	There is an illegal section type in the SECTION statement.
Display information	The name of the file and the line number where an error occurred
Action by user	Specify either CODE or DATA for the SECTION statement.

E0553030: unknown move operand 1.	
Description	There is an error in the first operand of the transfer directive.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the first operand.

E0553031: unknown move operand 2.	
Description	There is an error in the second operand of the transfer directive.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the second operand.

E0553032: unknown move operand 3.	
Description	There is an error in the third operand of the transfer directive.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the third operand.

E0553033: unknown push operand.	
Description	There is an error in the operand of the PUSH directive.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the operand.

E0553034: unknown pop operand.	
Description	There is an error in the operand of the POP directive.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the operand.

E0553035: displacement error.	
Description	There is an error in the description of the segment number for the transfer directive or displacement for the extended transfer directive.
Display information	The name of the file and the line number where an error occurred
Action by user	Specify a value from 0 to 3 for the segment number and from -128 to 127 for displacement.

E0553036: unknown port no.	
Description	There is an error in specification of port addresses of the IN and OUT directives.
Display information	The name of the file and the line number where an error occurred
Action by user	Specify a value from 0 to 255 for a port address.

E0553037: code format error.	
Description	There is an error in an instruction code of the CODE directive.
Display information	The name of the file and the line number where an error occurred
Action by user	Specify a value from 0 to 255 for an instruction code.

E0553038: unknown code. (*description of an unknown assembler directive*)	
Description	There is a description of an unknown assembler directive.
Display information	The name of the file, the line number where an error occurred, and the description of an unknown assembler directive
Action by user	Check the description of the assembler directive.

E0553039: reserved symbol. (*symbol name*)	
Description	A reserved word is included in a symbol name.
Display information	The name of the file, the line number where an error occurred, and the symbol name
Action by user	Change the symbol name which does not contain any reserved words.

E0553040: data format error.	
Description	There is an unknown description in the data section.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the data section.

E0553041: address(code) resolve error.	
Description	An undefined code label was referred to in the program area.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the definition of a label.

E0553042: address(code) format error.	
Description	There is an error in a description of an address which indicates the program area.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the address.

E0553043: address(data) resolve error.	
Description	An undefined data label was referred to in the program area.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the definition of a label.

E0553044: address(data) format error.	
Description	There is an error in a description of an address which indicates the data area.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the address.

E0553045: address resolve error.	
Description	An undefined label was referred to in the data section.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the definition of a label.

E0553046: data number error.	
Description	There is an error in a description of a numeral which follows the DATA keyword in the data section.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the numeral.

E0553047: section address format error.	
Description	There is an error in a description of an address which follows LOCATE in the SECTION statement.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the address.

E0553048: code label defined.	
Description	The definitions of labels are overlapped.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the definition of the label.

E0553049: section name defined.	
Description	The data section and code section names are overlapped.
Display information	The name of the file and the line number where an error occurred
Action by user	Specify the different name for a section.

E0553050: code section address base error.	
Description	The base address of the code section is outside the program area.
Display information	The name of the file and the line number where an error occurred
Action by user	Check specification of addresses in the code section.

E0553051: data label defined.	
Description	The definitions of labels are overlapped.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the definition of the label.

E0553052: section address overlapped.	
Description	The address ranges of sections are overlapped.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the address range of each section.

E0553053: data section address base error.	
Description	The base address of the data section is outside the data area.
Display information	The name of the file and the line number where an error occurred
Action by user	Check specification of addresses in the data section.

E0553054: .LINE line number is out of range.	
Description	The line number of the pseudo-directive .LINE is out of the range.
Display information	The name of the file and the line number where an error occurred
Action by user	Specify a value from 1 to 100,000 for a line number.

E0553055: macro definition number over.	
Description	The number of macros defined by {TC}define exceeded the upper limit (512).
Display information	The name of the file and the line number where an error occurred
Action by user	Specify the number of descriptions of {TC}define as less than 512.

E0553056: identifier string size over.	
Description	The character length of an identifier defined by {TC}define exceeded the upper limit (200).
Display information	The name of the file and the line number where an error occurred
Action by user	Check the character length of an identifier.

E0553057: function macro arguments number over.	
Description	The number of function-macro arguments exceeded the upper limit (31).
Display information	The name of the file and the line number where an error occurred
Action by user	Check the number of arguments.

E0553058: condition directives nesting over.	
Description	The number of nests for conditional inclusion (if, ifdef, ifndef, elif, or else) exceeded the upper limit (31).
Display information	The name of the file and the line number where an error occurred
Action by user	Specify the number of nests for conditional inclusion (if, ifdef, ifndef, elif, or else) as less than 31.

E0553059: statement nesting over.	
Description	The number of nests for control statements (if, elif, else, switch, while, during, or for) exceeded the upper limit (31).
Display information	The name of the file and the line number where an error occurred
Action by user	Specify the number of nests for control statements (if, elif, else, switch, while, during, or for) as less than 31.

E0553060: operator number in a statement over.	
Description	The number of operators used in a control statement exceeded the upper limit (31).
Display information	The name of the file and the line number where an error occurred
Action by user	Check the number of operators used in a control statement.

E0553061: line string size over.	
Description	The number of characters in a line exceeded the upper limit (2048).
Display information	The name of the file and the line number where an error occurred
Action by user	Specify the number of characters in a line as less than 2048.

E0553062: command line option -define define symbol not found.	
Description	No identifier is specified for the command-line option -define.
Display information	None
Action by user	Check that an identifier is defined.

E0553063: command line option -define text macro redefined. (*identifier name*)	
Description	A text macro was redefined with the command-line option -define. *If the command-line option "-allow_ text_macro _redefine" is specified, this error will not occur.
Display information	The name of the file and the line number where an error occurred
Action by user	Check that definitions of identifiers are not overlapped.

E0553064: command line option -define illegal define. (*identifier name*)	
Description	There was illegal specification of the command-line option -define.
Display information	The name of the identifier
Action by user	Check the specification of the command-line option -define.

E0553065: command line option -define macro definition number over.	
Description	The number of macros defined by the command-line option -define exceeded the upper limit (512).
Display information	None
Action by user	Specify the specified number of the command-line options -define as less than 512.

E0553066: command line option -define identifier string size over.	
Description	The character length of an identifier defined by the command-line option -define exceeded the upper limit (200).
Display information	None
Action by user	Check the character length of an identifier.

E0553067: command line option -define function macro arguments number over. (*identifier name*)	
Description	The number of function-macro arguments in the command-line option -define exceeded the upper limit (31).
Display information	The name of the identifier
Action by user	Check the number of arguments.

E0553068: command line option. (-output)	
Description	The invalid -output option is specified.
Display information	None
Action by user	Specify a character string for a parameter.

E0553069: command line option. (-define)	
Description	The invalid -define option is specified.
Display information	None
Action by user	Specify a character string for a parameter.

E0553070: command line option. (-inc_dir)	
Description	The invalid -inc_dir option is specified.
Display information	None
Action by user	Specify a character string for a parameter.

E0553071: out of section.	
Description	A code is described in the spot where no section has been defined.
Display information	The name of the file and the line number where an error occurred
Action by user	Describe the code below the start line of the code section or data section.

E0553074: not supported instruction.	
Description	A directive which is not supported by the V2 core version is described.
Display information	The name of the file and the line number where an error occurred
Action by user	Do not use directives which are not supported, otherwise specify 3 with the -core_version option.

E0553075: not supported register.	
Description	A register which is not supported by the V2 core version is described.
Display information	The name of the file and the line number where an error occurred
Action by user	Do not use registers which are not supported, otherwise specify 3 with the -core_version option.

E0553076: illegal operator. (*operator*)	
Description	An unavailable operator is described in [] in the control statement.
Display information	The name of the file, the line number where an error occurred, and operators
Action by user	Describe operators only available for the expression in the control statement.

E0553077: section end address is out of range.	
Description	The end address of a section exceeds 0xFFFFFFFF.
Display information	The name of the file and the line number where an error occurred
Action by user	Specify the end address of a section as 0xFFFFFFFF or lower.

E0553078: end statement not found.	
Description	There is no end statements for the control statements in the structured description.
Display information	The name of the file
Action by user	Describe the end statements for the control statements.

E0553079: invalid operand. (*operator operand*) *When a value of the right side of an operator is an unexpected operand invalid operand. (*operand operator*) *When a value of the left side of an operator is an unexpected operand	
Description	In the structured description, an unexpected operand is described for a value of the left side or the right side of an operator.
Display information	The name of the file, the line number where an error occurred, the operator where an error occurred, and the operand having an error
Action by user	Describe an operand which handles operators.

E0553080: section definition error.	
Description	Sections allocated in absolute and relocatable forms are mixed. Otherwise, there exists two or more sections allocated in relocatable form of which names are different.
Display information	The name of the file and the line number where an error occurred
Action by user	Define one of sections allocated in absolute form or relocatable form. Specify the same name for all sections allocated in relocatable form.

E0553081: invalid struct description. (*control statement*)	
Description	The control statement for the structured description is not the expected structure.
Display information	The name of the file, the line number where an error occurred, and the illegal control statement
Action by user	Check that the control structure described in the structured description is the same as that in section 4.4, Control Statements Available for the Structured Description.

E0553082: DSP unsupported operation. (*operand or operator*)	
Description	An operand or an operator which is unavailable for the DSP version specified with the -core_version option is described.
Display information	The name of the file, the line number where an error occurred, and the illegal operand or operator
Action by user	Do not use operators or operands which are not supported, otherwise specify 3 with the -core_version option.

E0553083: unknown pointer.	
Description	There is an error in a description of a pointer variable.
Display information	The name of the file and the line number where an error occurred
Action by user	Check the description of the pointer variable.

W0553084: MOVP instruction was inserted before JMP/JSR instruction.	
Description	A MOVP instruction has been added before a JMP or JSR instruction. Note: This warning is issued when the JMP or JSR instruction satisfies both of the conditions given below. • The branch destination is an address in the same segment. • The value of the lowest-order 12 bits of the address where the instruction is located is 0xFFE or 0xFFF.
Display information	The name of the file and the line number that the warning applies
Action by user	Check the address where the JMP or JSR instruction is located.

E0553085: command line option. (-code_section_start)	
Description	The description of the address where allocation is to start, which is specified with -code_section_start option, has an error, or the value is out of the applicable range.
Display information	None
Action by user	Check the value of the address where the allocation of code is to start.

E0553086: command line option. (-data_section_start)	
Description	The description of the address where allocation is to start, which is specified with -data_section_start option, has an error, or the value is out of the applicable range.
Display information	None
Action by user	Check the value of the address where the allocation of code is to start.

E0553087: An operator other than the logical AND/OR operator is used for the expression returning the boolean value.	
Description	An operator other than logical AND/OR has been used with an expression that may be evaluated as either true or false.
Display information	The name of the file and the line number where an error occurred
Action by user	Use the logical AND/OR operators to concatenate expressions for which the result may be true or false.

E0553090: command line option. (-macro_identify)	
Description	An invalid -macro_identify option is specified.
Display information	None
Action by user	Specify either FORWARD or EXACT.

E0553094: command line option. (-dwarf_spec)	
Description	An invalid -dwarf_spec option is specified.
Display information	None
Action by user	Specify any one of INITIAL, GENERIC, or RENESAS.

Revision History	DSPASM GREEN_DSP Structured Assembler User's Manual

Rev.	Date	Description	
		Page	Summary
1.00	Sep.09, 2016	—	First Edition issued
1.01	Jan.10, 2017	11	Table 2.11 -text_macro Command-Line Option Description about the cause of an assembler error was added.
		13	Table 2.17 -E Command-Line Option Description of combining -E command-line and -list command line options was added.
		13	Table 2.18 -cpuLittleEndian and -cpuBigEndian Command-Line Options Description of combining -cpuLittleEndian and -cpuBigEndian command-line options was added.
		25	4.1.3 A0 Register Bit Variables Description about the cause of an assembler error was added.
		26	Table 4.6 Pointer Variables Description of pointer variables for handling displacement was added.
		29	Table 4.8 Operators Description of operations when variables which cannot be handled by operators are used in the structured description was added.
		45	Table 4.14 for Control Statement Description of loop variables was deleted. Description of labels was added.
		55	Table 4.18 bset Instruction Description about the cause of an assembler error was added.
		55	Table 4.19 bclr Instruction Description about the cause of an assembler error was added.
		56	Table 4.20 btst Instruction Description about the cause of an assembler error was added.
		60	4.7 Automatic Generation of Constant Labels Description of automatically generated data destination for the structured description was added. Description of dedicated data section generation function was added.
		80	7.2.2 Operators in Control Statements Description about the cause of an assembler error was added.
		80	7.2.4 Variables That Cannot Be Handled by Operators Description of variables which cannot be specified on the right side or left side of the operator in the structured description was added.
		83	7.4 Differences of Code Generation Depending on the Core Version of DS Unnecessary description for core version 2 was deleted. Directives generated only in core version 3 were modified. Differences of code generation for relational operation was added.
		90 91	Table 10.2 Method for Counting Nesting Levels Table 10.3 Examples of Descriptions Where Nest Count of Conditional Inclusion Preprocessing Directive Exceeds Translation Limits Methods to count nests and description examples were added.
		95	11.2 Error Messages Error on -text_macro line option: E0553004: command line option. (-text_macro) Description and action were modified.

1.02	Sep.01, 2017	10	2.4 Command-Line Options for the DSPASM The following command line options were added to table 2.7, Command-Line Options. -code_section_start -data_section_start -no_debug_info
		14 15	2.4 Command-Line Options for the DSPASM Descriptions of the following command line options were added. -code_section_start -data_section_start -no_debug_info
		107	11.2 Error Messages The following error messages were added. E0553085: command line option. (-code_section_start) E0553086: command line option. (-data_section_start)
1.03	Dec.01, 2017	11 to 16	Statements about an assembly error being caused when any of the following command-line options is specified with no parameter were added. -format -output -text_macro -define -inc_dir -dsp -core_version -code_section_start -data_section_start
		37	Table 4.10 switch … case Control Statement A statement about the specification that writing two or more case labels having the same value leads to an assembler error was added.
		83	7.2.2 Operators in Control Statements A statement about the specification that operators cannot follow on from expressions for which the result may be true or false was added.
		87	Table 7.2 Character Sets Available in the Structured Description Special characters that are only usable in assembly statements were deleted.
		91	Table 8.1 Character Sets Available in the Assembly Description Special characters that are only usable in structured statements were deleted.
		91	Table 8.2, Characters and Symbols Available for the Label Name or Section Name, was added.
		110	11.2 Error Messages The following error message was added. E0553087: An operator other than the logical AND/OR operator is used for the expression returning the boolean value.
1.05	2022.12.01	6	Table 2.1 Operating Environment of the DSPASM A Windows 11 environment has been added.
		9-10	Table 2.7 Command-Line Options RL78_101_DSP, RL78_111_DSP, RL78_IAR_DSP, RL78_LLVM_DSP, and RL78_GCC_DSP have been added to DSP type of the -dsp option. The -label option has been added. The -macro_identify option has been added.

			The -dwarf_spec option has been added. The -code_execinstr option has been added.
		13	Table 2.15 -dsp Command-Line Option RL78_101_DSP, RL78_111_DSP, RL78_IAR_DSP, RL78_LLVM_DSP, and RL78_GCC_DSP have been added as DSP types.
		16-17	Table 2.23 -label Command-Line Option has been added. Table 2.24 - macro_identify Command-Line Option has been added. Table 2.25 - dwarf_spec Command-Line Option has been added. Table 2.26 - code_execinstr Command-Line Option has been added.
		70	Table 5.3 .public Pseudo-Directive has been added.
		85	8.3 Supplementary on Generating Assembly Codes A description about global symbols has been added.
		86	Table 9.4 Reserved Words for Assembly Descriptions (4) .public has been added.
		88	Table 10.1 Translation Limits on Preprocessing The number of descriptions of macro replacement preprocessing directives has been changed from 512 to 1,024.
		92	11.1 Formats of Error Messages In the error message format, file-name has been changed to file-path.
		92,105	11.2 Error Messages The description of the following message has been changed: E0553002: RL78_101_DSP, RL78_111_DSP, RL78_IAR_DSP, RL78_LLVM_DSP, and RL78_GCC_DSP have been added as DSP types. The following error messages have been added: E0553088: command line option. (-label) E0553090: command line option. (-macro_identify) E0553094: command line option. (-dwarf_spec)

DSPASM
FAA/GREEN_DSP Structured Assembler User's Manual

Publication Date: Rev.1.05 Dec 1, 2022

Published by: Renesas Electronics Corporation

DSPASM
FAA/GREEN_DSP Structured Assembler
User's Manual

Renesas Electronics Corporation

R20UT3911EJ0105