User's Manual

RENESAS

C/C++ Compiler Package for M16C Series and R8C Family V.6.00 Assembler, Optimizing Linkage Editor User's Manual

Renesas Electronics
www.renesas.com

Rev.1.00 Jan,2011

Preface

This manual explains how to use the assembler, and optimizing linkage editor for the M16C Series, R8C Family microcomputers. Please read this manual before using this system to fully understand the system. This system translates source programs written in assembly source programs into relocatable and absolute object programs for the M16C Series, R8C Family microcomputers.

Notes on Symbols: The following symbols are used in this manual.
Symbols Used in This Manual

Symbol	**Explanation**
< >	Indicates an item to be specified.
[ ]	Indicates an item that can be omitted.
...	Indicates that the preceding item can be repeated.
Δ	Indicates one or more blanks.
|	Indicates that one of the items must be selected.

This manual is intended for an IBM PC*1 compatible machine and Microsoft® Windows® XP operating system, Microsoft® Windows® Vista® operating system, or Microsoft® Windows® 7 operating system*2 that runs on other compatible machines.

Notes: 1. IBM is a registered trademark of International Business Machines Corporation
2. Microsoft® and Windows® are registered trademarks of Microsoft Corporation in the United States and other countries.
* All other company names and product names are trademarks or registered trademarks of corresponding companies.

Contents

Section 1 Overview

1.1 Configuration of Compiler

The configuration of the Assembler and the optimizing linkage editor for the M16C Series, R8C Family shown below.

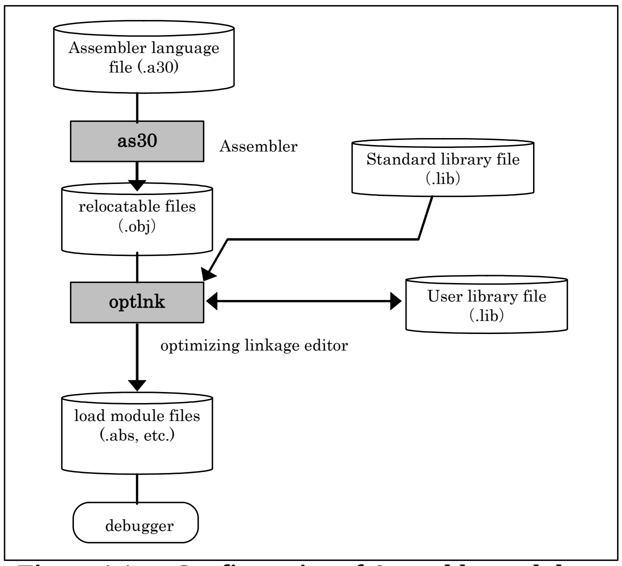

Assembler language file (.a30)
as30
Assembler
Standard library file (.lib)
relocatable files (.obj)
optlnk
User library file (.lib)
optimizing linkage editor
load module files (.abs, etc.)
debugger

Figure 1.1 Configuration of Assembler and the optimizing linkage editor

1.1.1 as30

as30 is an executable file of the assembler.
It converts the assembler source files (.a30) into object files (.obj).

as30 is comprised of the following programs:

- Assembler driver (as30)

  This program invokes the macro processor, structured preprocessor, and assembler processor in succession.

- Macro processor (mac30)

  It processes macro directive commands in the source file to generate an assembly language file.
  The assembly language files generated by the macro processor are removed after processing by the assembler processor is finished. The source files written by the user will in no case be modified.

- Structured processor (pre30)

  It processes the structured description commands in the source file to generate an assembly language file. The assembly language files generated by the structured preprocessor are removed after processing by the assembler processor is finished. The source files written by the user will in no case be modified. Specify the command option (-P) of as30 to invoke the structured preprocessor.

- Assembler processor (asp30)

  It converts the assembly language file preprocessed by the macro processor and structured preprocessor into an object file.

1.1.2 optlnk

optlnk is an executable file of the optimizing linkage editor.
It converts multiple object files (.obj) and library files (.lib) into an absolute file (.abs, etc.) or library file (.lib).

1.2 Rules for Specifying Options

The following describes the startup commands usable in the assembler and optimizing linkage editor.
Before using these commands, please see Section 7, “Environment Variables,” to confirm that the necessary environment variables have all been set.

1.2.1 Assembler (as30)

as30 is the startup command of the assembler.

[Command description format]

```
as30 [ Δ<option> …][ Δ<file name>[ Δ<option> …] …]
```

1.2.2 Optimizing Linkage Editor (optlnk)

optlnk is the startup command of the optimizing linkage editor.
Not just a link process, it also includes other functions listed below.

- Optimization when creating absolute files (.abs, etc.)
- Creation and editing of library files
- Conversion into Motorola S format files, Intel HEX format files, or binary files

[Command description format]

```
optlnk [ Δ<option> …][ Δ<file name>[ Δ<option> …] …]
        <option>: -<option>[=<suboption>][, …]
```

1.3 Contents of Upgrade and Migration Method

For details about the contents of upgrades from old versions, how to migrate the user application, and the precautions to take when migrating,
please see Appendix K, “Contents of Upgrade and Migration Method,” of C/C++ Compiler Package for M16C Series and R8C Family V.6.00 Compiler User’s Manual.

Section 2 Specifications of Assembler

2.1 Translation Limits of Assembler

Table 2.1 shows the translation limits of the assembler.

Table 2.1 Translation Limits of Assembler

No.	Item	Translation Limit
1	Number of characters in one line	8190
2	Symbol length	Number of characters in one line*
3	Number of symbols	Unlimited
4	Number of externally referenced symbols	Unlimited
5	Number of externally defined symbols	Unlimited
6	Maximum size for a section	0FFFFH or 0FFFFFH bytes
7	Number of sections	65265 (with debugging information) or 65274 (without debugging information)
8	File include	Nesting levels of 9
9	String length	Number of characters in one line*
10	Number of characters in a file name	Number of characters in one line*
11	Number of characters in an environment variable setting	2048 bytes
12	Number of macro definitions	65535

Note: *The limit may become a smaller value depending on the string length specified in the same line.

2.2 Character Set

You can use the following characters when writing an assembly program to be assembled by as30.

Table 2.1 Character Set

No.	Item	Character
1	Uppercase alphabets	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
2	Lowercase alphabets	a b s d e f g h i j k l m n o p q r s t u v w x y z
3	Numerals	0 1 2 3 4 5 6 7 8 9
4	Special characters	" # $ % & ' ( ) * + , - . / : ; [\ ] ^ _ | ~
5	Blank	(Space) (Tab)
6	New paragraph or line	(Carriage return) (Line feed)

Section 3 Assembler Language Description Rules

3.1 Precautions to Take when Writing a Program

When using as30, observe the following precautions as you write a program.

- Do not use the reserved words for labels, symbols, or bit symbols in the source program. The reserved words include the extensions "IF", "ENDIF", etc.

- Strings consisting of the directive commands of as30 with the period removed may be used for names without causing an error. However, use of these strings is not recommended, because some of them may affect processing of as30.

- System labels (strings that begin with ..) will not cause an error providing that they are written in the source program by the user. However, use of these strings is not recommended, because they may be used for the future extension of as30.

3.2 Rules for Writing a Program

3.2.1 Character Sets

The character sets shown in Section 2, "Specifications of Assembler" can be used to write a source program.

3.2.2 Reserved Words

as30 handles the same strings as the assembler directives and mnemonics as reserved words. Since the reserved words have special functionality, they cannot be used for label or symbol names in the source program. Note also that the reserved words are not discriminated between uppercase ad lowercase letters . "ABS" and "abs" are interpreted as the same reserved word.
The reserved words include the following:

(1) Directive assemble commands

All directive assemble commands explained in this manual and all character strings that begin with one period are the reserved words.

(2) Mnemonic

All of the M16C series, R8C family mnemonics are the reserved words.

(3) Register and flag names

All of the M16C series, R8C family register and flag names are the reserved words.

(4) Operators

All of the operators and structured operators described in this manual are the reserved words.

(5) Structured description commands

All of the structured description commands described in this manual are the reserved words.

(6) System labels

The labels generated by the assembler are referred to as the system labels. All names that begin with two periods (..) are handled as system labels.

3.2.3 Names

Any name can be defined and used in an assembly language file.
Names are classified into the following types, each of which has a different permissible scope of description.

Table 3.1 Types of Name

Type of name	Description
Label	A name that has an address as value.
Symbol	A name that has a constant as value.
Bit symbol	A name that has a constant (bit position) and address as value. Each bit in an 8-bit long memory location can be assigned a specific name for discrimination.
Section	The name of a section defined by .SECTION directive command.
Macro	The defined name of a macro.
Location symbol	Indicates the start address of the operation part of a line that contains the location symbol '$'.

Rules for Writing a Name

- The number of characters comprising a name conforms to the number of characters per line that is mentioned in Section 2.1, "Translation Limits of Assembler"
- Alphanumeric characters, underscore (_), and dollar sign ($) can be used for a name.
- No digits can be used at the top of a name.
- Names are discriminated between uppercase and lowercase letters. "LAB" and "lab" are handled as different names.

Note: No names can be used that are the same as the reserved words. If this restriction is neglected, program behavior cannot be guaranteed.

(1) Labels

- Label names conform to "Rules for Writing a Name".
- To define a label, be sure that a colon (:) is added at the end of the name.
- Labels can be defined in a section.

- A label name can be specified when reserving storage by a directive command.

```
Exsample) flags:  .BLKB    1
          work:   .BLKD    1
```

- A label name can be written at any place in the source line.

```
Exsample) name1:
             _name:
           sym_name:
```

- To reference a label, write its name in the operand of the instruction used.

```
Exsample) JMP    sym_name
```

(2) Symbols

- Symbol names conform to "Rules for Writing a Name".
- Numeric values must be determinate at assemble execution time.
- Symbols can be defined inside or outside a section.
- Use the directive command ".EQU" that defines a numeric value.

```
Exsample)  value1  .EQU  1
           value2  .EQU  2
```

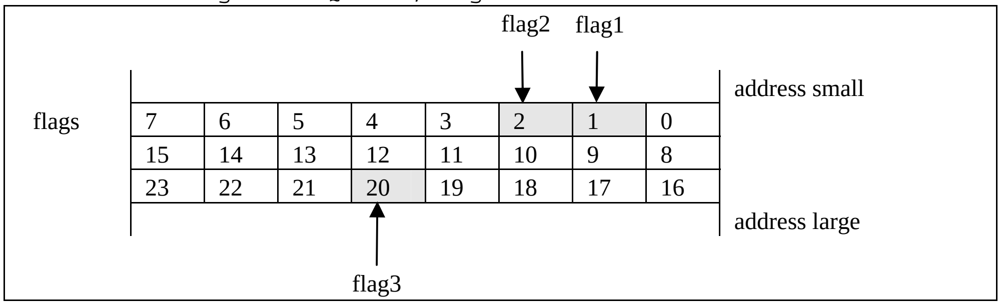
- To reference a symbol, write its name in the operand of the instruction used.

```
Exsample)  MOV.W   R0,value1
           value3  .EQU    value2+1
```

(3) Rules for Writing Bit symbol

- Bit symbol names conform to "Rules for Writing a Name".
- For the numeric values that specify a bit position, specify a value that is determinate at assemble execution time.
- Bit symbols can be defined inside or outside a section.
- Use the directive command ".BTEQU" that defines a bit symbol.

```
Exsample)  flags  .EQU    400H
           flag1  .BTEQU  1,flags
           flag2  .BTEQU  2,flags
           flag3  .BTEQU  20,flags
```



- Bit symbols can be written in the operand of a 1-bit operation instruction used.

```
Exsample)  BCLR  flag1
           BCLR  flag2
           BCLR  flag3
```

(4) Sections

- Section names conform to "Rules for Writing a Name".
- For details about sections, refer to the directive command “.SECTION” in Section 9.3, "Link Control Directive Commands".

(5) Macros

- Macro names conform to "Rules for Writing a Name".
- For details about macros, refer to the directive command ".MACRO" in Section 9.6, "Macro Directive Commands".

(6) Rules for writing a location symbol

- Write in the operand of a mnemonic.
- A location symbol can be written in a term of an expression.
- A location symbol can be written in a structured description statement.

```
Exsample)     JMP.B  $+5
              [label] = $
              [label] = $+1
```

Note: When writing an address that is offset by a location symbol in the mnemonic of a branch instruction, be careful that optimization will not be applied to any mnemonics from that position up to the jump address.

3.3 Method for Writing a Line

3.3.1 Typesof Line

as30 processes the source program one line at a time. Lines are classified by the content written in each as follows:

(1) Directive command lines

- This is the line where a directive command of as30 is written.
- Only one directive command can be written in one line.
- A comment can be written in a directive command line.

Note: Nor can a directive command and a mnemonic be written in one and the same line.

(2) Assembler source lines

- This is the line where a mnemonic is written.
- A comment can be written in an assembler source line.
- A label name can be written at the top of an assembler source line.

Note: Two or more mnemonics cannot be written in one line.
Nor can a directive command and a mnemonic be written in one and the same line.

(3) Label definition lines

- This is the line where only a label name is written.

(4) Comment lines

- This is the line where only a comment is written.

(5) Null lines

- This is the line that contains only a space, tab or new-line code.

3.3.2 Rules for Writing a Lines

(1) Line separation

Lines are separated by a new-line character, with a range of characters from the one that immediately follows the new-line character up to the next new-line character constituting one line.

(2) Line length

See Table 2.1, "Translation Limits of the Assembler".
Note that any characters exceeding the maximum number of characters are not processed.

Note: When writing a line, make sure that each line is written within the designed range of characters.

3.3.3 Rules for Writing a Directive Command Line

- Be sure that a space or tab is written between a directive command and its operand.
- To write multiple operands, be sure that a comma (,) is written between one operand and another.
- A space or tab can be written between an operand and a comma.
- Some directive commands do not require writing an operand.
- Directive commands can be written beginning with the top of a line.
- A space or tab can be written at the top of a directive command line.
- To write a comment in a directive command line, write a semicolon (;) next to the directive command and operand and then a comment in places following the semicolon. Comments are output to an assembler list file.
- A space or tab can be written between the operand of a directive command and a comment.

Note: as30 processes the content written in places following a semicolon (;) all as a comment. The assembler does not generate code for any mnemonics and directive commands written in places following a semicolon. Be careful with the position at which a semicolon (;) is written. If a semicolon (;) is enclosed in double-quotes (") or single-quotes ('), as30 does not assume it to be the beginning character of a comment.

```
Exsample)       .SECTION ram,DATA
                .ORG     00H
         sym    .EQU     0
         work:  .BLKB    1
```

3.3.4 Rules for Writing an Assembler Source Line

For details on how to write mnemonics, see Software Manual.
Here, explanation is made of the rules for writing an assembler source line processable by as30.

- Be sure that a space or tab is written between a mnemonic and its operand.
- To write multiple operands, be sure that a comma (,) is written between one operand and another.
- A space or tab can be written between an operand and a comma.
- Some mnemonics do not require writing an operand.
- Mnemonics can be written beginning with the top of a line.
- A space or tab can be written at the top of an assembler source line.
- To define a label in an assembler source line, be sure that a label name is written in places preceding the mnemonic.
- Be sure that a colon is written immediately after the label name of label definition.
- A space or tab can be written between a label name and mnemonic.

- To write a comment in an assembler source line, write a semicolon (;) next to the mnemonic and operand and then a comment in places following the semicolon.
- Comments are output to an assembler list file.
- A space or tab can be written between the operand of a mnemonic and a comment.

3.3.5 Rules for Writing a Label Definition Line

- Be sure that a colon (:) is written immediately after a label name.
- Do not write anything between a label name and a colon (:).
- Label names can be written beginning with the top of a line.
- A space or tab can be written at the top of a line.
- Comments are output to an assembler list file.
- A space or tab can be written between a label and a comment.

```
Exsample)     start:
              label: .BLKB  1
              main:  nop
              loop:
```

3.3.6 Rules for Writing a Comment Line

- Be sure that a semicolon (;) is written at the top of a comment.
- Comments can be written following a directive command line, assembler source line, and label definition line.
- A space or tab can be written at the top of a comment line.
- Any characters can be written in a comment.

```
Exsample)    ; Comment line
             MOV.W     #0,A0    ; Comment can be written in other lines too.
```

Note: as30 does not generate code for any mnemonics and directive commands written in places following a semicolon (;). Be careful with the position at which a semicolon (;) is written. If a semicolon (;) is enclosed in double-quotes (") or single-quotes ('), as30 does not assume it to be the beginning character of a comment.

3.3.7 Rules for Writing a Null Line

- A line that contains no characters can be written as necessary to improve the readability of the source program or for other purposes.
- No characters other than a space, tab, return, and line feed can be written in a null line.

```
Exsample)   loop:
             ;
             JMP    loop
```

3.4 Line Concatenation

- If a line contains a '\\', the next line is concatenated into this line at the position where the '\\' is written.
- Comments can be written in a line that contains a '\\'. However, no comments are output to the result of concatenation.
- If an error occurs in a line that contains a '\\', the error is output for the last line concatenated.
- An example of how to write a line concatenation and the result of concatenation are shown below.

Note: Make sure the maximum number of characters in a line derived as the result of concatenation does not exceeds the number of characters stipulated in Table 2.1, "Translation Limits of the Assembler". However, the number of characters here does not include spaces and tabs at the tops of lines concatenated. Be aware that if a '\' is written immediately after a 2-byte code character, it may be erroneously recognized as a '\\'.

```
Exsample1)
       .BYTE   1,\\
               2, \\
               3    \\
               ,4
Result of concatenation:)
       .BYTE   1,2, 3      ,4

Example2)
       .BYTE   1,\\  ;comment
               2,    ;comment \\
               3     ;comment
Result of concatenation:2)
       .BYTE   1,2, ;comment
               3     ;comment

Example3)
       .BYTE   1,\\
               2,\\
               3,    \\
               4
Result of concatenation:3)
       .BYTE   1,2,3,      4
```

3.5 Operands

3.5.1 Types of Operands

In a mnemonic or directive command, an operand can be written that indicates the subject to be operated on by that instruction. There are following types of operands.

Note: Some instructions do not have the operand. For information on whether an instruction has the operand, see a section in which rules for writing each instruction are described.

(1) Numeric values

Integers and floating-point numbers can be written.

(2) Names

Label and symbol names can be written.

(3) Expressions

Expressions that have a numeric value and a name in their terms can be written.

(4) Strings

Characters and character strings are handled as ASCII code.

3.5.2 Rules for Writing an Operand

The position at which an operand is written

Be sure that a space or tab is written between an operand and an instruction that has the operand.
The sections that follow describe rules for writing each operand.

3.5.3 Numeric Values

Following types of numeric values are supported, which can be written in the source file.
Note that the internally representable range of numeric values is from –2147483648 to 2147483647.

- Binary numbers
- Octal numbers
- Decimal numbers
- Hexadecimal numbers
- Floating-point numbers

(1) Binary numbers

Use digits 0 to 1 to write a binary number and add the letter 'B'or 'b' as a suffix.

```
    Exsample)    10010001B
                 10010001b
```

(2) Octal numbers

Use digits 0 to 7 to write an octal number and add the letter 'O'or 'o' as a suffix.

```
    Exsample)    60702O
                 60702o
```

(3) Decimal numbers

Use digits 0 to 9 to write a decimal number.

```
    Exsample)    9423
```

(4) Hexadecimal numbers

Use digits 0 to 9 and letters A to F and a to f to write a hexadecimal number and add the letter 'H' or 'h' as a suffix.

If the value begins with an English alphabet, add a zero (0) as a prefix.

```
    Exsample)    0A5FH
                 5FH
                 0a5fh
                 5fh
```

(5) Floating-point numbers

Floating-point numbers can only be written in the operands of control instructions ".FLOAT" and ".DOUBLE".

Floating-point numbers cannot be written in expressions.

Any value in the range representable by floating-point numbers can be written, as given below.

FLOAT (32 bits long): $1.17549435 * 10^{-38}$ to $3.40282347 * 10^{38}$

DOUBLE (64 bits long): $2.2250738585072014 * 10^{-308}$ to $1.7976931348623157 * 10^{308}$

```
    Rules for Writing)     (mantissa)E(exponent)
                           (mantissa)e(exponent)
```

Exsample) 3.4E35 ;$3.4*10^{35}$

3.4e-35 ;$3.4*10^{-35}$

-.5E20 ;$-0.5*10^{20}$

5e-20 ;$5.0*10^{-20}$

3.6 Expressions

Expressions created using a numeric value, a name, and an operator in combination can be written.

- A space or tab can be written between the operator and the numeric value.
- The operator can be written as a combination of multiple operators.
- To write an expression as a symbol value, make sure the value of the expression is determinate at assemble time.
- The values resulting from operation on expressions are in the range –2147483648 to 2147483647.
- Floating-point numbers cannot be written in the terms of expressions.

Note: When the result of an operation exceeds the range –2147483648 to 2147483647, no out-of-range error is assumed.

3.7 Operators

The operators that can be written in the source program of as30 are listed below.Unary operators

- Unary operators

Table 3.2 Unary Operators

Operator	Function
+	Handles the subsequent value as a positive value.
-	Handles the subsequent value as a negative value.
~	Handles logical negation of the subsequent value.
SIZEOF	Handles the size (in bytes) of the section specified in the operand as a value.
TOPOF	Handles the start address of the section specified in the operand as a value.

For SIZEOF and TOPOF, write a space character or tab between the operator and its operand.
Example: SIZEOF program

- Binary operators

Table 3.3 Binary Operators

Operator	Function
+	Adds the left value and right value together.
-	Subtracts the right value from the left value.
*	Multiplies the left value and right value together.
/	Divides the left value by the right value.
%	Yields the remainder from the division of the left value by the right value.
>>	Bit-shifts the left value to the right as many times as the right value.
<<	Bit-shifts the left value to the left as many times as the right value.
&	Logically ANDs the left and right values bitwise.
|	Logically ORs the left and right values bitwise.
^	Exclusive-ORs the left and right values bitwise.

- Conditional operators

Conditional operator can only be written in the operand of the ".IF" or ".ELIF" directive.

Table 3.4 Conditional Operators

Operator	Function
>	Tests if the value of the left value is greater than that of the right.
<	Tests if the value of the left value is less than that of the right.
>=	Tests if the value of the left value is greater than or equal to that of the right.
<=	Tests if the value of the left value is less than or equal to that of the right.
==	Tests the equivalence of left and right values.
!=	Tests the negated equivalence of left and right values.

- Precedence designation operator

Table 3.5 Precedence Designation Operator

Operator	Function
( )	An operation enclosed within ( ) takes precedence. If multiple pairs of parentheses are used in an expression, the left pair is given precedence over the right pair. Parentheses can be nested.

3.8 Operation Priority of Expressions

Expressions written in the operand are operated on according to the priority given below, and the numeric values resulting from the operation are handled as a value.

- Operation is performed in order of priority of operators, beginning with the one that has the highest priority. The priority of operators is shown in the table below.
- Operators with the same priority are operated on in the order they are written, from left to right.
- The operator enclosed in ( ) has the highest priority.

Table 3.6 Order of Expression Evaluation

Precedence	Operator Type	Operator
1	Precedence designation operator	()
2	Unary operator	+, -, ~, SIZEOF, TOPOF
3	Binary operator 1	*, /, %
4	Binary operator 2	+, -
5	Binary operator 3	>>, <<
6	Binary operator 4	&
7	Binary operator 5	|, ^
8	Conditional operator	>, <, >=, <=, ==, !=

3.9 Strings

Strings can be written in the operands of some directive commands. Characters consisting of 7-bit long ASCII code can be used to write strings. To write a string in the operand, enclose it in single or double-quotes unless otherwise noted.

```
Exsample)   "string"
            'string'
```

3.10 Outline of Mnemonic Description

For details about the rules for writing assembler mnemonics, see M16C Series, R8C Family Software Manual for each MCU used.

(1) Size specifiers

Specify the size of data to be operated on by a mnemonic (B, W, or L). Always be sure to specify it.

(2) Jump distance specifiers

Specify the distance to the target address of jump and subroutine call instructions. It need not normally be specified.
If the operand involves indirect addressing, write a jump distance specifier. If it is omitted, an error results.

(3) Instruction format specifiers

Specify the form of op-code. If instruction formats (Z, Q, Z, or S) differ, the code lengths of instruction op-codes and operands differ. It need not normally be specified.

(4) Addressing mode specifiers

Specify the addressing mode of operand data. In as30, part of a mnemonic that specifies the address range of relative addressing is referred to as the addressing mode specifier.

```
Exsample)   ":16" and ":8" are the addressing mode specifiers
            MOV.W    work1:16[SB],work2:8[SB]
```

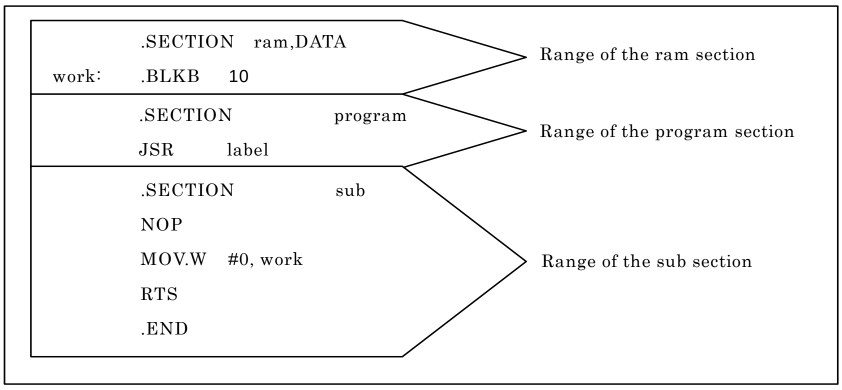
Section 4 Programming

4.1 Section

The executable instruction and data areas in the object files output by the assembler respectively comprise a section. Delimitation of sections is defined as follows:

- An interval from the line where a directive command ".SECTION" is written to a line immediately preceding the one in which the next directive command ".SECTION" is written
- An interval from the line where a directive command ".SECTION" is written to a line immediately preceding the one in which a directive command ".END" is written

Note that if multiple sections with the same name exist, they are concatenated into one section.



```
        .SECTION    ram,DATA
work:   .BLKB       10
        .SECTION            program
        JSR         label
        .SECTION            sub
        NOP
        MOV.W       #0, work
        RTS
        .END
```

4.1.1 Types of Sections

as30 outputs relocatable information in units of sections. Sections are classified into the following by the instruction and section declaration written inside a section. Classification is determined by a combination of attribute and type, as in the case of "DATA type section with the absolute attribute".

- Absolute attribute section (attribute)
- Relative attribute section (attribute)
- CODE type section (type)
- DATA type section (type)
- ROMDATA type section (type)

(1) Absolute attribute sections

- This is the section whose code location address is determined at assemble time.
- The addresses in a section become absolute values at assemble time.
- The values of the labels specified in an absolute section are absolute.

- To make a section assume the absolute attribute, use the directive command ".ORG" to specify addresses in a line next to the one in which the directive command ".SECTION" is written.

```
Exsample)    .SECTION   program,CODE
             .ORG       1000H
```

(2) Relative attribute sections

- This is the section whose code location address is not determined at assemble time.
- The addresses in a section become relocatable values at assemble time.
- The values of the labels defined in a relative section are relocatable.
- To make a section assume the relative attribute, do not use the directive command ".ORG" to specify addresses in a line next to the one in which the directive command ".SECTION" is written.

```
Exsample)    .SECTION   program,CODE
```

(3) CODE type sections (program area)

- This is the section where a program is written.
- All instructions, except the directive commands to reserve storage, can be written.
- The CODE type sections are mapped to the ROM area.

```
Exsample)    .SECTION   program,CODE
```

(4) DATA type sections (variable data areas)

- This is the area where memory with variable contents are located.
- The directive commands to reserve storage can be written.
- The DATA type sections are mapped to the RAM area.

```
Exsample)    .SECTION   mem,DATA
```

(5) ROMDATA type sections (fixed data areas)

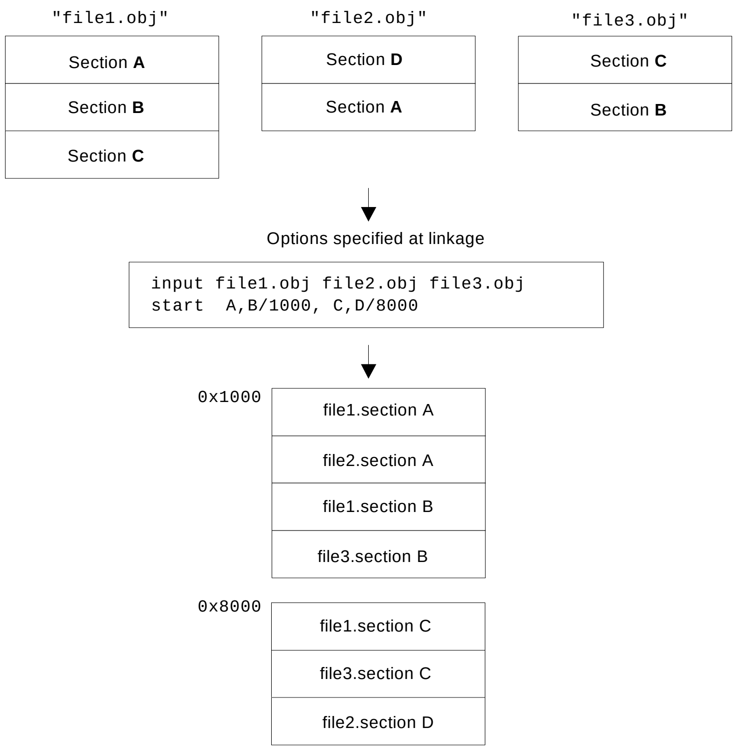
- This is the area where fixed data other than a program are written.
- The directive commands to set data can be written.
- All instructions, except the directive commands to reserve storage, can be written.
- The ROMDATA type sections are mapped to the ROM area.

```
Exsample)    .SECTION   const,ROMDATA
```

4.1.2 Linking Sections

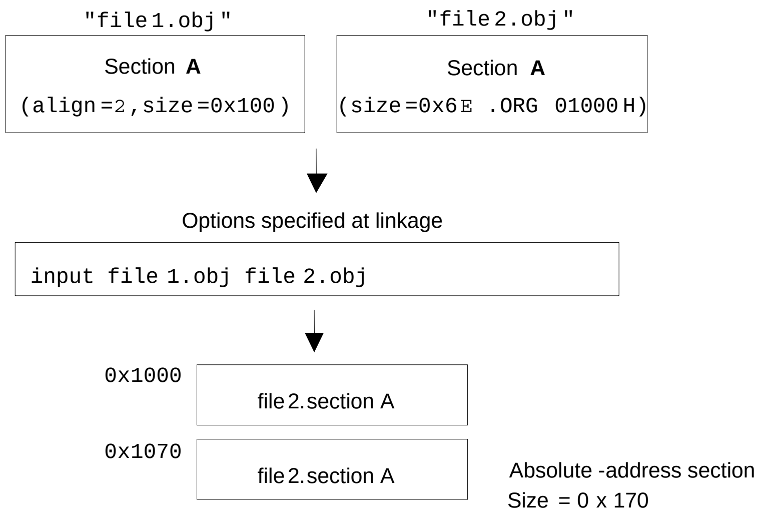
The optimizing linkage editor links the same sections within input relocatable files, and allocates addresses specified by the start option.

(1) The same section names in different files are allocated continuously in the order of file input.



```
input  file1.obj  file2.obj  file3.obj
start   A,B/1000,  C,D/8000
```

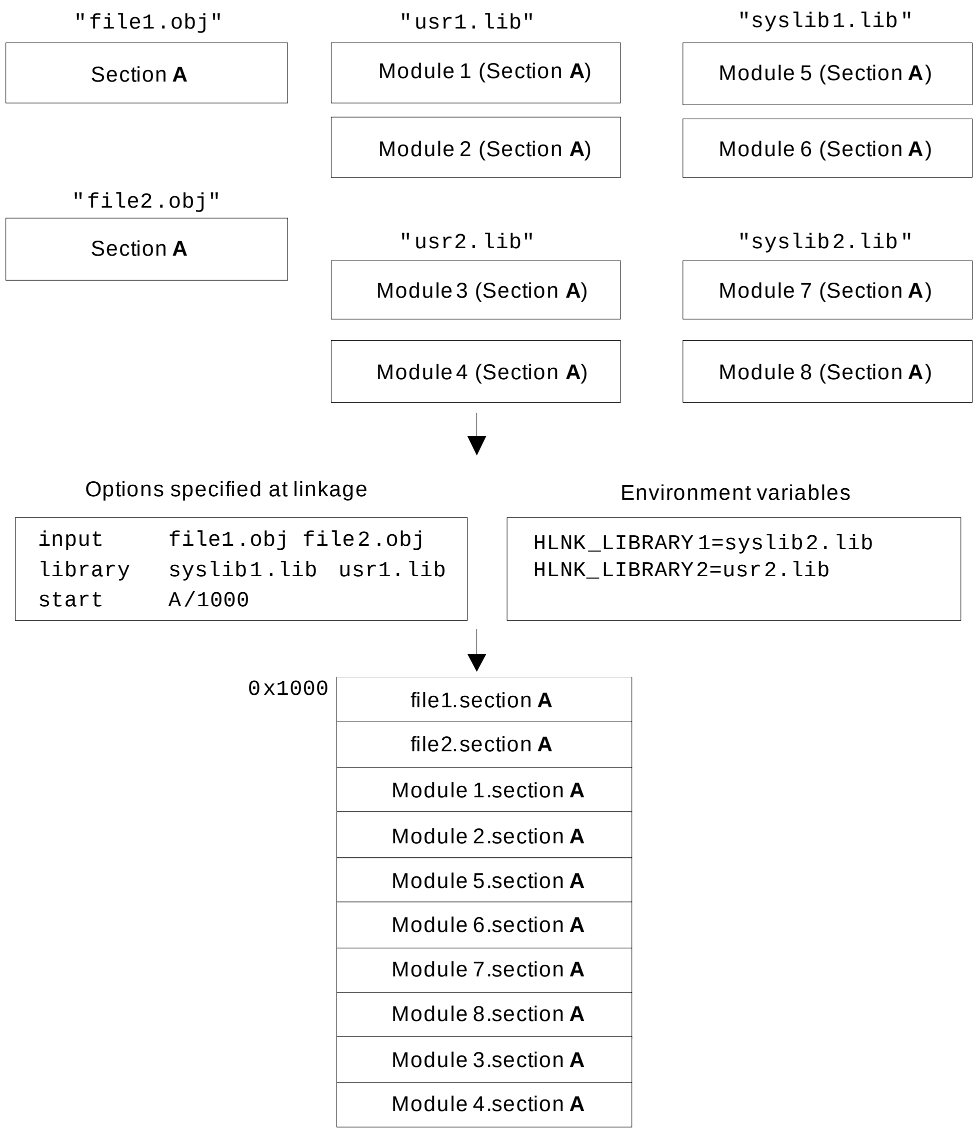
(2) When sections with the same name include both absolute-address and relative-address formats, relative-address sections are linked following absolute-address sections.



```
input  file 1.obj  file 2.obj
```

(3) Rules for the order of linking sections with the same name are based on their priorities as follows.

- Order specified by the input option or input files on the command line
- Order specified for the user library by the library option and order of input of modules within the library
- Order specified for the system library by the library option and order of input of modules within the library
- Order specified for libraries by environment variables (HLNK_LIBRARY1 to HLNK_LIBRARY3) and order of input of modules within the library



```
input      file1.obj file2.obj
library    syslib1.lib  usr1.lib
start      A/1000
```

```
HLNK_LIBRARY1=syslib2.lib
HLNK_LIBRARY2=usr2.lib
```

4.2 Labels and Symbols

4.2.1 Attributes

Label and symbols (I include the bit symbol) are classified into the following four attributes.

- Global
- Local
- Relocatable
- Absolute

Attribute is decided by combining the definition and the reference to the label and symbols.

	Global	Local	Relocatable	Absolute
Global	---	×	O	O
Local	×	---	O	O
Relocatable	O	O	---	×
Absolute	O	O	×	---

Example: The global attribute can be used in combination with the relocatable attribute or absolute attribute.

(1) Global

- The labels and symbols specified with the directive command ".GLB" become global labels and global symbols, respectively.
- The bit symbols specified with the directive command ".BTGLB" become global bit symbols.
- The names defined in a file that are specified as global become referencible from external files.
- The names not defined in a file that are specified as global become external reference labels, symbols, or bit symbols that reference the names defined in external files.
- Information on names with the global attribute are output to the object file.

(2) Local

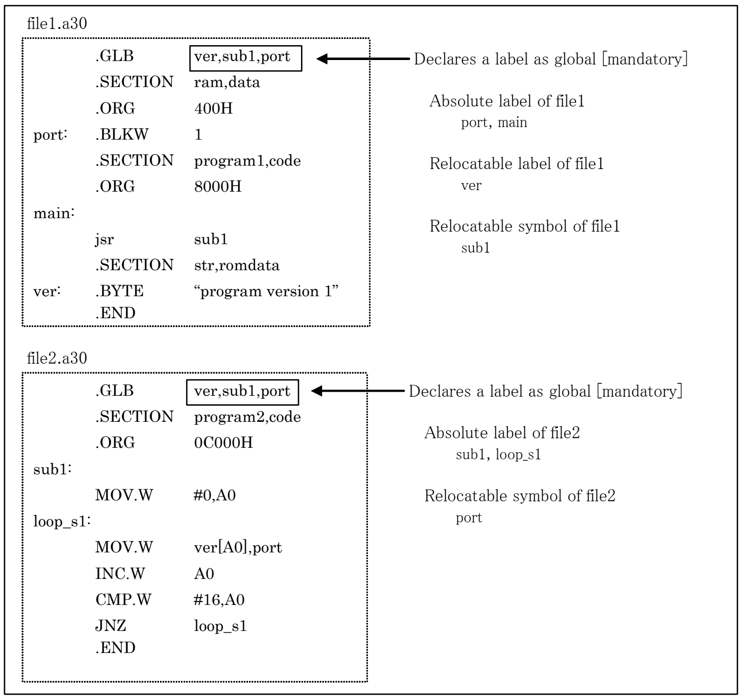
- Names not specified with the directive command ".GLB" or ".BTGLB" are all local.
- Local names are referenced in only the same file as they are defined.
- Local names are local, so that the same label names as theirs can be used in another file.
- Information on only the local labels and local symbols that are defined in relative attribute sections are output to the object file. However, if assembled after specifying a command option (-S or -SM), information on all local labels and local symbols are output to the object file.

(3) Relocatable

- The labels defined in a relative attribute section assume the relocatable attribute.
- Externally referenced labels, symbols, and bit symbols assume the relocatable attribute.

(4) Absolute

- The labels defined in an absolute attribute section assume the absolute attribute.
- The symbols and bit symbols defined with constants assume the absolute attribute.



file1.a30

```
        .GLB        ver,sub1,port
        .SECTION    ram,data
        .ORG        400H
port:   .BLKW       1
        .SECTION    program1,code
        .ORG        8000H
main:
        jsr         sub1
        .SECTION    str,romdata
ver:    .BYTE       "program version 1"
        .END
```

Declares a label as global [mandatory]
Absolute label of file1
port, main
Relocatable label of file1
ver
Relocatable symbol of file1
sub1

file2.a30

```
        .GLB        ver,sub1,port
        .SECTION    program2,code
        .ORG        0C000H
sub1:
        MOV.W       #0,A0
loop_s1:
        MOV.W       ver[A0],port
        INC.W       A0
        CMP.W       #16,A0
        JNZ         loop_s1
        .END
```

Declares a label as global [mandatory]
Absolute label of file2
sub1, loop_s1
Relocatable symbol of file2
port

4.2.2 Determination of Values

(1) Absolute attribute

The values of label and symbols that have the absolute attribute are determined at assemble execution time.

(2) Relocatable attribute

The values of label and symbols that have the relocatable attribute are determined at link time. If, as a result of linking, any jump instruction or addressing mode determined by the assembler exceeds the specifiable range, a warning is output.

4.2.3 Symbol Definition by a Command Option

as30 allows symbols to be defined using a command option (–D) at program startup time. This symbol definition function can be used in combination with the conditional assembly function, etc. For details, see the section in which the conditional assembly function is described.

4.3 References to Include Files

as30 allows an include file to be loaded in any line of the source program. This function may be used to improve the readability of the program.

(1) Rules for writing an include file

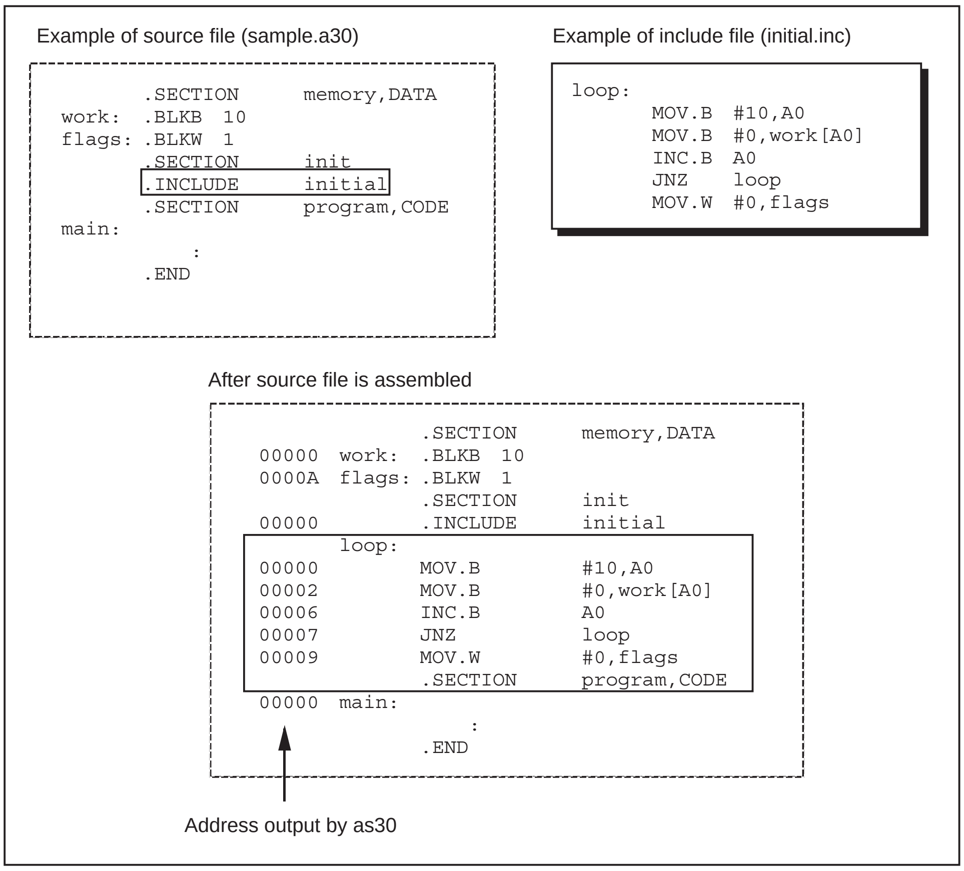
To write an include file, follow the rules for writing the source program.

Notes: Do not write the directive command ".END" in an include file.

(2) Loading an include file

Write the file name to be loaded in the operand of the directive command ".INCLUDE". All contents of the include file are read into the position of this line.

Example:



Example of source file (sample.a30)

```
         .SECTION      memory,DATA
work:    .BLKB  10
flags:   .BLKW  1
         .SECTION      init
         .INCLUDE      initial
         .SECTION      program,CODE
main:
               :
         .END
```

Example of include file (initial.inc)

```
loop:
        MOV.B  #10,A0
        MOV.B  #0,work[A0]
        INC.B  A0
        JNZ    loop
        MOV.W  #0,flags
```

After source file is assembled

```
                   .SECTION      memory,DATA
00000   work:      .BLKB  10
0000A   flags:     .BLKW  1
                   .SECTION      init
00000              .INCLUDE      initial
        loop:
00000             MOV.B          #10,A0
00002             MOV.B          #0,work[A0]
00006             INC.B          A0
00007             JNZ            loop
00009             MOV.W          #0,flags
                   .SECTION      program,CODE
00000   main:
                          :
                   .END
```

4.4 Selection of Code Optimization by as30

as30 selects the shortest possible code from the M16C series, R8C family addressing modes.
The assembler performs code optimization when the following conditions apply.

- The jump distance specifier is omitted (normally omitted).
- The instruction format specifier is omitted (normally omitted).
- The addressing mode specifier is omitted.
- Combination of the above.

(1) When the jump distance specifier is omitted

as30 performs optimum code selection when all of the following conditions are met.

- The operand is written with one label or written with an expression that includes one label.

  Label
  Label + determinate value at assemble time
  Label – determinate value at assemble time
  Determinate value at assemble time + label

- Labels of the operand are defined in the same section in the same file.
- The section in which the instructions are written and the section in which the operand labels are defined both have the absolute attribute and these sections are written in the same file.

Optimization rules

- Unconditional jump instructions

  The shortest jumpable instruction is selected from the jump distances '.A', '.W', '.B', or '.S'.
  The size '.S' is selected only when the jump instruction and the label for the target address exist in the same section.

- Subroutine call instructions

  The shortest jumpable instruction is selected from the jump distances '.A' or '.W'.

- Conditional jump instructions

  The jump distance '.B' or alternative instruction is generated.

Notes: The source line information in the list file consists of the source lines that were output directly as written. The code information part contains the code for alternative instructions output to the file. For the "ADJNZ" and "SBJNZ" instructions, as30 performs optimization of jumps equivalent to conditional jump instructions.

(2) When the instruction format specifier is omitted

- as30 performs optimum code selection for the mnemonics which have had the instruction format specifier omitted.
- as30, if the instruction format specifier is omitted, determines addressing mode before it selects an instruction format.

(3) When the addressing mode specifier is omitted (selection of SB relative or FB relative)

If the addressing mode specifier is omitted, as30 performs optimum code selection when the following conditions are met.

- For addressing with displacement, the value of displacement is determined at assemble execution time. In this case, optimum addressing mode is selected.
- The directive command ".SB" or ".FB" is defined. In this case, 8-bit SB relative addressing mode (hereafter referred to as SB relative) or 8-bit FB relative addressing mode (hereafter referred to as FB relative) is selected depending on condition.

The following shows the necessary conditions for each of these addressing modes to be selected.

(a) Conditions for SB relative addressing modes to be selected

SB relative is selected when one of the following conditions is met:

- The operand value is determinate at assemble execution time and this value is in the SB relative selectable range. The SB relative selectable range means that the operand value is in the range of address 0 to 0FFFFH and in the range +0 to +255 relative to the 16-bit register (SB).
- The symbols declared by the directive command ".SBSYM" are written in the op-code.

```
Example where SB relative is selected:
      .SB      100H
      symbol   .EQU   108H
      ABS.B    symbol
```

Notes: When using SB relative addressing, be sure to set the SB register value with the directive command ".SB". If the SB register value is defined by an expression that is indeterminate at assemble time, optimization is not performed.

Selection of SB relative addressing mode by a bit instruction addressing instruction

- If the mnemonic has a short form in its instruction format, short-form SB relative is selected.
- If the mnemonic does not have a short form in its instruction format, 16-bit SB relative addressing mode is selected.

(b) Conditions for FB relative addressing modes to be selected

FB relative is selected when one of the following conditions is met:

- The symbols declared by the directive command ".FBSYM" are written in the op-code.
- The expressions given below that include the symbols declared by the directive command ".FBSYM" are written in the op-code.

  (symbol) + determinate value at assemble time
  (symbol) – determinate value at assemble time
  Determinate value at assemble time + (symbol)

Notes: When using FB relative addressing, be sure to set the FB register value with the directive command ".FB".

(4) When the addressing mode specifier is omitted (selection of address register indirect)

If the addressing mode specifier is omitted, as30 performs optimum code selection when the following conditions are met.

- For addressing with displacement, the value of displacement is determined at assemble execution time and the determined value is 0. In this case, optimum addressing is selected.

```
Example where address register indirect is selected:
     ABS.B   symbol[A0]   ; Selects address register indirect.
     ABS.B   symbol:8[A0] ; Selects address register relative.
```

4.5 SB Register Offset Address Specification

In as30 programming, it is possible to write offset address specification relative to the SB register value.

- The address value specified by the directive command ".SB" plus a specified offset value is the target value to be operated on.
- Code is generated for SB relative addressing mode.
- Specify the offset in an operand in which SB relative addressing mode can be written.
- Labels, symbols, or numeric values can be used to write the offset.

```
Exsample)
       sym1       .EQU     1200H
       .SECTION   P
       .SB        1000H
       MOV.B      #0,sym1[SB]
       MOV.B      #0,sym1[-SB]
       .END
```

4.6 Special Page Vector Table

Special page jump

In the M16C family assembly language, by writing a "JMPS" mnemonic it is possible to cause a special page jump using the special page vector table.

Special page subroutine

In the M16C family assembly language, by writing a "JSRS" mnemonic it is possible to make a special page subroutine call using the special page vector table.

Special page vector table

The special page vector table is outlined below.

- The special page vector table is allocated to the addresses 0FFE00H through 0FFFDBH.
- One vector table consists of 2 bytes.
- Each vector table is assigned one special page number.

- Special page numbers are decremented from 255 to 254 and so on every 2 bytes beginning with the address 0FFE00H.

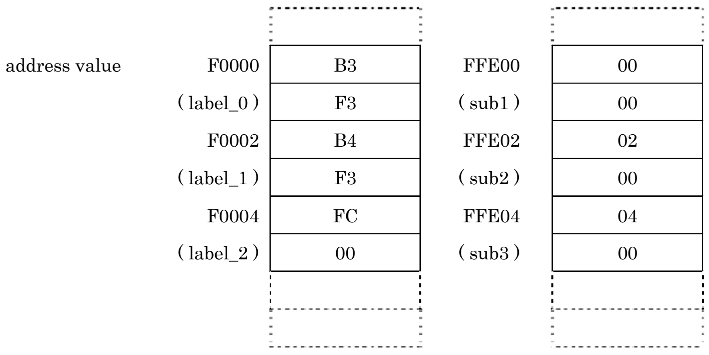
For details about the special page vector table, see Software Manual.
In this manual, the method for setting up and referencing the special page vector table are described.
To generate the special page vector table automatically, see the chapter in which the assembler directive ".SVECTOR" is described.

4.6.1 Setting Up the Special Page Vector Table

In the special page vector table, store the 16 low-order bits of the special page addresses.

- Define a section that is used exclusively for the special page vector table.
- Define absolute addresses with the directive command ".ORG".
- Be sure that the addresses set here are even addresses within the range from 0FFE00H to 0FFFDBH.
- Store the 16 low-order bits of the special page addresses in ROM using the directive command ".WORD".

```
Exsample)
                .SECTION  sp_vect,ROMDATA
                .ORG      0FFE00H
      sub1:     .WORD     label_0 & 0FFFFH ; Special page number 255
      sub2:     .WORD     label_1 & 0FFFFH ; Special page number 254
      sub3:     .WORD     label_2 & 0FFFFH ; Special page number 253
                ;
                .ORG      0FFFDAH
      sub238:   .WORD     label_238 & 0FFFFH ; Special page number 18
```



4.6.2 Referencing Special Page Vector Table

There are following two methods to look up the special page vector table.

- Specify the address of a special page vector table.
- Specify a special page number.

Description rules

- To specify the address of a special page vector table, be sure to write '\' at the beginning.
- To specify a special page number, be sure to write '#' at the beginning.

```
Specifying the addresses of special page vector tables)
            .SECTION  p
       main:
            JSRS   \sub1
            JSRS   \sub2
            JSRS   \sub3
            .SECTION  special
            .ORG   0F0000H
       label_0:
            MOV.B  #0,R0H
            RTS
       label_1:
            MOV.B  #0,R0L
            RTS
       label_2:
            JMP    main
            .END

Specifying special page numbers)
            .SECTION  p
       main:
            JSRS   #255
            JSRS   #254
            JSRS   #253
            ;
```

4.7 Macro Functions

The macro functions usable in as30 are described below. There are two types of macro functions that as30 supports.

- Macro function

  By defining a macro with the directive commands ".MACRO" and ".ENDM" and then invoking the defined macro, it is possible to use the macro function.

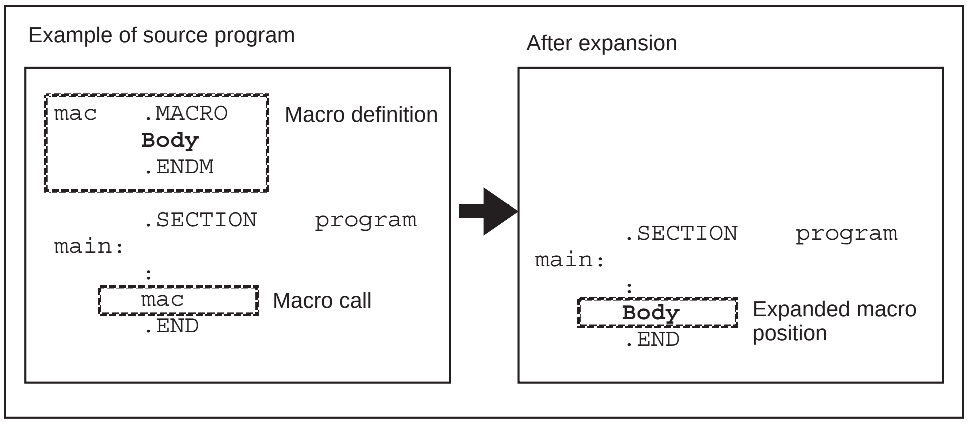
- Repeat macro function

  By writing the directive commands ".MREPEAT" and ".ENDR", it is possible to use the repeat macro function.

Each macro function is described below.

4.7.1 Macro Function

- The macro function can be used by a macro invocation of a macro-defined macro name.
- The macro function cannot be used by a macro definition alone.
- Macro definitions and macro invocations have the following relationship.



Macro definition

- For a macro definition, use the directive command ".MACRO" to define a collection of one line or more of instructions in one macro name.
- Macro names and macro parameters are discriminated between uppercase and lowercase letters.
- Use the directive command ".ENDM" to indicate the end of a macro definition.
- A line enclosed by the directive commands ".MACRO" and ".ENDM" is called the macro body.
- Parameters can be defined in a macro definition.
- A recursive macro definition is accepted.
- Macros can be nested up to 65,535 levels including macro definitions and macro invocations.
- A macro with the same name as another one can be redefined.

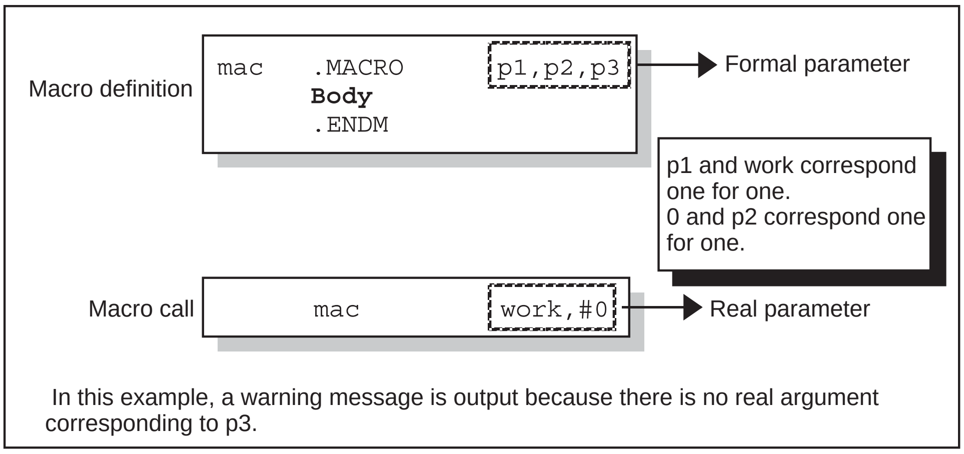
- A macro definition can be written outside the scope of a section.
- Any instructions writable in a source program can be written in the macro body.
- Macro parameters (80 pcs. or less) can be written.
- Macro local labels can be written for up to a sum total of 65,535 pcs. in one assembler source file.

Macro local labels

- The labels declared by the directive command ".LOCAL" become a macro local label.
- Macro local labels can be used in only a macro definition.
- Label names by which macro local labels are declared are local, so that a label with the same name as one of those macros can be written outside the scope of the macro.
- To use any label as a macro local label, declare it to be a macro local label before defining it.

Macro call

- By writing a macro name defined with the directive command ".MACRO" in the program, it is possible to make a macro invocation.
- Code for the macro body is generated by a macro invocation.
- Forward references to a macro name (i.e., writing a macro name that is defined in a line next to the macro invocation line) are not accepted. Be sure that a macro definition is written in a line preceding the invocation line.
- External references to a macro name (i.e., writing a macro name that is defined in another file) are not accepted. To invoke one and the same macro from multiple files, define it in an include file and include that file in the executable file.
- Arguments corresponding to the macro-defined parameters can be written.


Macro definition
mac .MACRO p1,p2,p3
Body
.ENDM
Formal parameter
p1 and work correspond one for one.
0 and p2 correspond one for one.
Macro call
mac work,#0
Real parameter
In this example, a warning message is output because there is no real argument corresponding to p3.

4.7.2 Repeat Macro Function

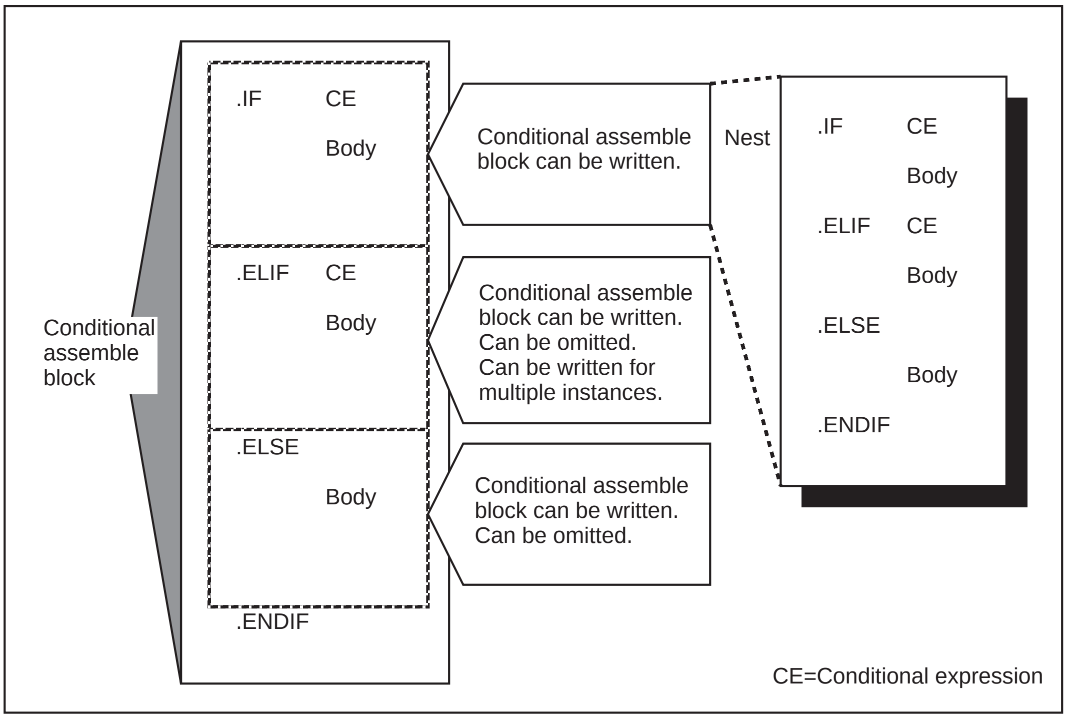
- A body enclosed by the directive commands ".MREPEAT" and ".ENDR" is repeatedly expanded a specified number of times in and beneath a specified line
- A repeat macro is expanded in the line where it is defined.
- A label can be written in a repeat macro definition line.

Notes: This label is not a macro name. There are no macro invocations for a repeat macro.

4.8 Conditional Assembly Function

Conditional assembly refers to controlling whether or not to convert the source lines into machine language according to the given condition. No code is generated for the lines that were not assembled as judged from the condition.

Structure of the conditional assembly block


Conditional assemble block
.IF CE
Body
.ELIF CE
Body
.ELSE
Body
.ENDIF
Conditional assemble block can be written.
Conditional assemble block can be written. Can be omitted. Can be written for multiple instances.
Conditional assemble block can be written. Can be omitted.
Nest
.IF CE
Body
.ELIF CE
Body
.ELSE
Body
.ENDIF
CE=Conditional expression

Example for executing conditional assembly

Shown below is an example where three kinds of messages are selected before assembling.

The assembler source file name in this example is "sample.a30".

```
Example of a source)
             .SECTION  outdata,ROMDATA,ALIGN
      .IF    TYPE==1
             .BYTE  "PROTO TYPE"
      .ELIF  TYPE>1
             .BYTE  "MASS PRODUCTION TYPE"
      .ELSE
             .BYTE  "DEBUG MODE"
      .ENDIF
             .END

Example of command input 1)
      > as30 sample.a30 -Dtype=1

Assembly result 1)
      .SECTION  outdata,ROMDATA,ALIGN
      .BYTE  "PROTO TYPE"
      .END

Example of command input 2)
      > as30 sample -Dtype=2

Assembly result 2)
      .SECTION  outdata,ROMDATA,ALIGN
      .BYTE  "MASS PRODUCTION TYPE"
      .END

Example of command input 3)
      > as30 sample -Dtype=0

Assembly result 3)
      .SECTION  outdata,ROMDATA,ALIGN
      .BYTE  "DEBUG MODE"
      .END
```

For the case where conditions are written in the source file

Here is how to set a value for "type" in the assembler source file.

```
TYPE   .EQU   1
       .SECTION  outdata,ROMDATA,ALIGN
.IF           TYPE==1
       .BYTE  "PROTO TYPE"
.ELIF  TYPE>1
       .BYTE  "MASS PRODUCTION TYPE"
.ELSE
       .BYTE  "DEBUG MODE"
.ENDIF
       .END
```

Section 5 Assembler Options

5.1 Rules for Specifying Command Parameters

Always enter a command at the command prompt.
Only one period can be used in a file name.

5.2 Composition of the Command Line

Enter the following information on the command line.

Program name

The name of the program used.

Command parameters

The command parameter includes the command options that indicate by a symbol the target file name to be processed by an activated program and the functionality of the program.
The command parameter contains the following information:

- File name

  The name of the file to be processed by a program.

- Command options

  Added at program startup time in order to make use of the basic functionality of each program.

5.3 Rules for Entering Information on Command Line

Follow the rules described below to enter the necessary information on the command line to activate each program of as30.

- Number of characters per command line

  The number of characters that can be entered on the command line is 2,048 characters (bytes) or less. Depending on the working environment of as 30 (type of OS), the number of characters usable on the command line may be limited to less than the above.

- Be sure to write a space between the program name to be activated and the file name.

  Be sure to write a space between the file name and a command option and between each command option.

- The rules for writing a file name are subject to restrictions imposed by the OS other than the above. For details, see the user's manual of the OS used.

- The period (.) usable in a file name can be inserted in only one place.

- For rules on file extension (characters following the period), check the method for activating the respective programs.

5.4 Rules for Specifying Command Parameters

The order in which command parameters are specified

- Command options and assembler source file names can be specified in any order.

Assembler source file names (mandatory)

- Be sure to specify at least one assembler source file name.
- A path can be specified for the assembler source file name.
- Up to 80 assembler source file names can be specified.
- The files with the extension ".a30" can have their extensions omitted.

Notes: Of the multiple assembler source files specified, if there is any assembler source file that contains an error, the remaining other files are not processed.

Command options

- Command options can be omitted.
- Multiple command options can be specified.
- For some command options, a string or numeric value can be specified.

Notes: Do not write a space or tab between a command option and a string or numeric value.
It is only the command options of as30 that the user can disable command options. This feature cannot be used in the other programs to be activated.

Examples:

```
> as30   sample   -L              --- (1)
> as30   sample   -S              --- (2)
> as30   sample   -L   -S   --L   --- (3)
```

(1) Specify a command option "-L"

(2) Specify a command option "-S"

(3) Specify a command option "-L"

5.5 Assembler Command Options

5.5.1 Source Options

Table 5.1 Source Options

No.	option	Description	Dialog menu
1	-I	Specify a directory in which include files are searched.	Assembly<Source> [Show entries for:] Include file directories
2	-D	Define a symbol.	Assembly<Source> [Show entries for:] Defines

Specify an include file search directory

-I

Format: -I<path name>

Description: Searches the specified directory for the directive command ".INCLUDE" that are written in the source file.

This option can be specified only once. If it is specified twice or more, the content of the last option specified is valid.

For the order in which include files are searched, see the chapter where the directive command ".INCLUDE" is described.

Example:

```
> as30 -I\work\include sample.a30
```

Searches the work\include directory for the specified include files.

Remarks: Specify a directory path name after "-I" in succession.

No space or tab can be written between the option and the directory path name.

Define a symbol

-D

Format: -D<symbol name>=<numerical value>[:<symbol name>=<numeric value>. . .]

Description: Sets a value for a symbol.

The values set here are handled as absolute values.

The symbols defined by this option are handled the same way as are the symbol definitions written in the assembler source file. This means that if a symbol definition with the same name as the one defined here is written in the assembler source file, the symbol defined here is redefined at that position in the source file.

The symbols defined by this option are processed in the same way as for the symbol definitions made at the beginning in the source program. However, they are not output to the assembler list file.

Examples:

```
> as30 -Dname=1 sample.a30                    (1)
> as30 -Dname=1:symbol=1 sample.a30           (2)
> as30 -Dname=1 sample1.a30 sample2.a30       (3)
```

(1) Set the symbol 'name' to 1.
(2) Set the symbols 'name' and 'symbol' to 1.
(3) Define the symbol 'name' for sample1 and sample2 files.

Remarks: Do not write a space or tab between the command option and symbol name.

Multiple symbols can be defined with values. To define multiple symbols¸ use a colon to separate each entry written, as in "-D(symbol name)=(numeric value):(symbol name)=(numeric value) ".

No space or tab can be written before and after the colon.

5.5.2 Object Options

Table 5.2 Object Options

No.	option	Description	Dialog menu
1	-S[M]	Output local symbol information	Assembly<Object>
2	-finfo	Generate inspector information	Assembly<Object>
3	-goptimize	Output additional information for inter-module optimization	Assembly<Object>
4	-N	Do not output debug information	Assembly<Object>
5	-P	Convert structured description command	Assembly<Object>
6	-M	Convert structured description command into byte-type mnemonic	Assembly<Object>
7	-O	Specify directory for output files	Assembly<Object>

Output local symbol information

-S

Format: -S [M]

Description: Outputs local symbol information to the object file.

When the letter "M"is added to this option, system label information also is output to the object file.

To perform symbolic debugging using local symbols, specify this option before assembling.

Examples:

```
> as30 -S sample.a30                    (1)
> as30 -SM sample.a30                   (2)
```

(1) Output the local symbol information of sample.a30 to sample.obj.
(2) Output the system label information and local symbol information of sample.a30 to sample.obj.

Notes: Check the symbol information in the linkage list to confirm the symbol and label information.

The subroutine start label name written in the operand of the directive command ".INSF" and the subroutine name written in the operand of the directive command ".CALL" are always output regardless of whether this option is specified.

Generate inspector information

-finfo

Format: -finfo

Description: Outputs each information generated by the "-finfo" option of NC30 or the inspector information written with assembler directives to the object file.

Example:

```
> as30 -finfo sample.a30
```

Remarks: Because characters are case-sensitive, use lowercase letters for all characters of this option when specifying it.

Notes: When projects are built in Renesas integrated environment, this option is specified by default.

Output additional information for inter-module optimization

- goptimize

Format: -goptimize

Description: Outputs additional information for inter-module optimization.

The files which have had this option specified become the target of inter-module optimization when linked.

Example:

```
> as30 -goptimize sample.a30
```

Notes: If this option is specified, even when the jump distance specifier is written in unconditional jump instructions or subroutine call instructions, the files become the target of inter-module optimization by optlnk.

When this option is specified, as30 generates code for the jump distance of unconditional jump instructions, subroutine call instructions, conditional jump instructions, add & conditional jump instructions, and subtract & conditional jump instructions in sizes given below.

Table 5.3 File Format Specifiers

mnemonic	Jump distance specifier	Jump distance
	.S	PC[*1] + 2 <= operand <= PC[*1]+8
JMP	.B	PC[*1] - 126 <= operand <= PC[*1] + 127
	.W	PC[*1] - 32766 <= operand <= PC[*1] + 32767
JSR	.W	PC[*1] - 32766 <= operand <= PC[*1] + 32767
GEU/C, GTU, EQ/Z.N, LTU/NC, LEU, NE/NZ, PZ	---	PC[*1] - 126 <= operand <= PC[*1] + 127
LE, O, GE, GT, NO, LT	---	PC[*1] - 125 <= operand <= PC[*1] + 128
ADJNZ	---	PC[*1] - 125 <= operand <= PC[*1] + 128
SBJNZ	---	PC[*1] - 125 <= operand <= PC[*1] + 128

*1 PC denotes the start address of an instruction.

Do not output debug information

-N

Format: -N

Description: Does not output debug information to the object file.

This helps to reduce the size of the object file.

Example:

```
> as30 -N sample.a30
```

Notes: Source line level debugging cannot be performed in the absolute files created from the object files that were generated after specifying this option.

Convert structured description command

-P

Format: -P

Description: Processes the structured description commands written in the assembler source file.

Example:

```
> as30 -P -LS sample.a30
```

Process the structured description commands in the assembler source file and output the expanded part to the assembler list file.

Remarks: When structured description commands are used, be sure to specify this option.

Convert structured description command into byte-type mnemonic

-M

Format: -M

Description: Processes the variables whose types are not determined in structured description commands as having byte type.

Examples:

```
> as30 -P -M sample.a30
> as30 -M -P sample.a30
```

Remarks: Specify this option simultaneously with the command option "-P".

If this option is not specified, variables whose types are not determined are processed as having word type.

Specify directory for output files

-O

Format: -O<directory name>

Description: Specifies the directory to which the object files, assembler list files, and assembler error tag files generated by the assembler are output.

The directory name specified here can include a drive name. It can also be specified with a relative path.

Examples:

```
> as30 -Oc:\work\asmout sample.a30            (1)
> as30 -O..\tmp sample.a30                    (2)
> as30 -Oc:\work\asmout sample -L -T          (3)
```

(1) Output the object files to the "\work\asmout" directory on drive C.
(2) Output the object files to the tmp directory that is the parent directory of the current directory.
(3) Output the object files, assembler error tag files, and assembler list files to the "\work\asmout" directory on drive C.

Remarks: No space or tab can be written between this option and the directory name.

5.5.3 List Options

Table 5.4 List Options

No.	option	Description	Dialog menu
1	-L	Generate assembler list file.	Assembly<List>
2	-H	Do not output header information of assembler list file.	Assembly<Source>

Generate assembler list file

-L

Format: -L [C | D | I | M | S]

Description: Generates an assembler list file in addition to the object file.

The generated list files have the extension ".lst".

If any directory is specified with the command option "-O", assembler list files are generated in the specified directory.

Example:

```
> as30 -LM sample.a30
```

Remarks: A file format specifier 'C', 'D', 'I', 'M', or 'S' can be specified in this option.

No space or tab can be written between the file format specifier and "-L".

Multiple file format specifiers can be specified at the same time.

File format specifiers can be specified in any order.

Table 5.5 File Format Specifiers

Format specifier	Function
C	Output line concatenations to the list file directly as are.
D	Output information prior to .DEFINE replacements to the list file.
I	Output lines whose conditional assemble condition was false to the assembler list file.
M	Output expanded lines of macro descriptions to the assembler list file.
S	Output expanded lines of structured description commands to the assembler list file.

Do not output header information of assembler list file

-H

Format: -H

Description: Suppresses output of assembler list file header information.

Example:

```
> as30 -L -H sample.a30
```

Remarks: Specify this option simultaneously with the command option "-L".

5.5.4 Turning Options

Table 5.6 Turning Options

No.	option	Description	Dialog menu
1	-A	Evaluate mnemonic operand.	Assembly<Tuning >
2	-PATCH[6N]TA -PATCH[6N]TAn	Generate code to avoid precautions for 3-phase motor control timer functions.	Assembly<Tuning >

Evaluate Mnemonic Operand

-A

Format: -A

Description: Outputs a warning for mnemonics that accept specification of both immediate and address value when the mnemonic concerned does not have a '#' written in it indicating that the operand is an immediate.

Examples: Example of a source description:

```
        .section     prg,code
        MOV.W        0,400H
        .end
```

Example of an output list file when "-A" is specified:

```
1                                   .section   prg,code
2       00000       73FF00000004    MOV.W      0,400H
sample.a30(2) : A1207 (W) Addressing is described by the numerical value
3                                   .end
```

Notes: A warning is output when the operand is a numeric value except labels or a symbol whose value is determinate at assemble time.

Generate code to avoid precautions for 3-phase motor control timer functions

-PATCH[6N]_TA／-PATCH[6N]_TAn

Format: -PATCH_TA, -PATCH_TAn, -PATCH6N_TA, -PATCH6N_TAn

Description: Generate code to avoid precautions for 3-phase motor control timer functions.

Avoidance code is generated only when a value is written by a MOV instruction (in words) to the address indicated by the timer A1-1 register (TA11), timer A2-1 register (TA21), or timer A4-1 register (TA41). (The above address applies to only the values that are determinate at assemble time.)

Example 1: Example of a source description:

```
.section    prg,code
MOV.W       #7E, TA11
.end
```

Example of an output list file when "-PATCH_TA" is specified:

```
1                                  .section     prg,code
2   00000        75CF42037E00      MOV.W        #7E, TA11
                 75CF42037E00      ; This is a line which AS30 output.
3                                  .end
```

-> The same MOV instruction written is generated as avoidance code.

Example 2: Example of a source description:

```
.section    prg,code
MOV.W       #7E,TA11
.end
```

Example of an output list file when "-PATCH_TA2" is specified:

```
1                                  .section     prg,code
2   00000        75CF42037E00      MOV.W        #7E,TA11
                 0404
                 75CF42037E00      ; This is a line which AS30 output.
3                                  .end
```

-> As many NOP instructions as specified by "n" and the same MOV instruction written are generated as avoidance code.

Remarks: Decimal numbers from 0 to 99 can be specified for "n" in "-PATCH_TAn".

Be sure to use uppercase letters to specify this option.

Table 5.7 Addresses for Which Avoidance Code is Generated

Specified option	**Addresses for which avoidance code is generated**
-PATCH_TA, -PATCH_TAn	Address 342H for TA11, address 344H for TA21, address 346H for TA41
-PATCH6N_TA, -PATCH6N_TAn	Address 1C2H for TA11, address 1C4H for TA21, address 1C6H for TA41

Notes: This option cannot be specified simultaneously with the "-R8C" option.

For details about notes, see MAEC TECHNICAL NEWS No. M16C-95-0302.

5.5.5 Other Options

Table 5.8 Other Options

No.	option	Description	Dialog menu
1	-.	Stop message output to the screen.	Assembly<Other>
2	-C	Display command lines passed to mac30, pre30, and asp30 by as30.	Assembly<Other>
3	-F	Fix file names indicated by ..FILE to the source file name.	Assembly<Other>
4	-V	Display version numbers of all programs.	Assembly<Other>
5	-subcommnad= <file name>	Load command line from files.	Assembly<Other> [User-defined options]
6	-T	Generate assembler error tag files.	Assembly<Other> [User-defined options]
7	-X	Activate external program using tag file as argument.	Assembly<Other> [User-defined options]

Stop message output to the screen

-.

Format: -.

Description: Does not output messages to the screen which would otherwise be output when as30 performs processing.

Nevertheless, error messages, warning messages, and the messages asserted by the directive command ".ASSERT" are output.

Example:

```
> as30 -. sample
```

Examples of output messages:

When -. is specified:

```
>as30 -. sample.a30
sample.a30(2): A2225 (E) Section type is not appropriate
```

When -. is not specified:

```
>as30 sample.a30
M16C Series and R8C Family Assembler system Version 6.00.00
Copyright (C) 1995 (1996 - 2010) Renesas Electronics Corporation and
Renesas Solutions Corp. All rights reserved.

 (sample.asm )
macro processing now

assembler processing now
sample.a30(2): A2225 (E) Section type is not appropriate

TOTAL ERROR(S)          00001
TOTAL WARNING(S)        00000
TOTAL LINE(S)           00007   LINES
CODE             0000003(00003H) program
```

Display command lines passed to mac30, pre30, and asp30 by as30

-C

Format: -C

Description: Permits the command options added when as30 activated mac30, pre30, or asp30 to be confirmed on the screen.

Example 1: When "-C" is specified, information is displayed as follows (beginning with the line next to copyright message "All Rights Reserved"):

```
>as30 -C -L sample

( sample.a30 )
mac30.exe -L -rREV.F sample.a30
macro processing now

asp30.exe -finfo -no_utl -G -L sample.m30
assembler processing now
```

Example 2: When specified in combination with the option to stop message output to the screen, information is displayed as follows:

```
>as30 -. -C -L sample

( sample.a30 )
mac30.exe -L -rREV.F sample.a30
asp30.exe -finfo -no_utl -G -L sample.m30
```

Fix file names indicated by ..FILE to the source file name

-F

Format: -F

Description: Fixes the file name expanded by the directive command "..FILE" to that of the assembler source file specified from the command line.

Example:

```
> as30 -F sample.a30
```

The file name expanded by the directive command "..FILE" written in the file "include.inc" that is included by the sample.a30 assembler source file becomes "sample".

If this option is not specified, the file name expanded by the directive command "..FILE" becomes "include".

Display version numbers of all programs

-V

Format: -V

Description: Displays the version numbers of all programs included with as30 and finishes processing.

All other parameters on the command line are ignored.

No object files are output.

Example:

```
> as30 -V
```

Remarks: Specify this option only.

Inputs command line specifications from a file

-subcommand

Format: -subcommand<file name>

Description: To specify the "-subcommand" option, specify the startup options of the assembler in a subcommand file.

The syntax in a subcommand file is the same as that of the command line.

Examples: Content of the subcommand file opt.sub:

```
-L -H
```

Subcommand specification:

```
>as30 -subcommand=opt.sub sample.a30
```

Interpretation by the assembler:

```
>as30 -L -H sample.a30
```

Notes: The option "-subcommand" cannot be specified in subcommand files.

Generate assembler error tag files

-T

Format:	-T
Description:	Generates an assembler error tag file when assembler errors or warnings occurred.

The file is output in a format suitable for the editor's tag jump function.

Even when this option is specified, no files will be generated if there are no errors.

If errors occurred at assemble time, no object files are generated. If only warnings occurred, object files are generated.

The error tag file name is derived from the assembler source file name, with its extension changed to ".atg".

Example:

```
> as30 -T sample.a30
```

If errors occur, a "sample.atg" file is generated.

Activate external program using tag file as argument

-X

Format:	-X<external program>
Description:	Generates an assembler error tag file and then activates the executable program specified after "-X".

If errors occur, an assembler error tag file is generated regardless of whether "-T" is specified and the assembler error tag file is opened in the specified program.

Example:

```
> as30 -Xedit sample.a30
```

Remarks: No space or tag can be written between this option and the program name.

5.5.6 CPU Options

Table 5.9 CPU Options

No.	option	Description	Dialog menu
1	-R8C	Generate code for the R8C series (Memory space, address 0H to address 0FFFFH).	CPU<CPU type> [Generate code for R8C family (ROM less than 64KB)]
	-R8CE	Generate code for the R8C series (Memory space, address 0H to address 0FFFFFH).	CPU<CPU type> [Generate code for R8C family (ROM 64KB or more)]
2	-R8Cxx	Generate code to avoid precautions for clock synchronous serial with chip select (SSU) or I^2C bus interface (IIC)	CPU<CPU type> [Generate code for R8C family (ROM less than 64KB)] Limitations on R8C/14, 15, 16, 17 avoided

Generate code for the R8C series

-R8C/-R8CE

Format: -R8C

-R8CE

Description: Generates code appropriate for the R8C series.

Table 5.10 CPU Options

Specified option	Memory space
-R8C	Address 0H to address 0FFFFH
-R8CE	Address 0H to address 0FFFFFH

Example:

```
> as30 -R8C sample.a30
```

Remarks: Use uppercase letters to specify this option.

Notes: When this option is specified, the symbol constant setting option "-D_R8C_=1" is added.

This option cannot be specified simultaneously with "-PATCH[6N]_TA" or "-PATCH[6N]_TAn".

When this option is specified, the directive command ".SVECTOR" cannot be used.

When this option is specified, the special page jump instruction (JMPS) and special page subroutine call instruction (JSRS) cannot be used.

Generate code for the R8C series

-R8Cxx

Format: -R8C<group-name>

Description: Generates code to avoid precautions for clock synchronous serial with chip select (SSU) or I^2C bus interface (IIC). (This applies only when the above register address values are determinate at assemble time.)

When a group name that has the SSU or IIC function is specified, a message "R8C/xx group in information file 'r8ctiny.txt' is used" is output.

The other functions are the same as the option "-R8C" is specified.

Examples:

```
> as30 -R8C14 sample.a30
```

Generate code to avoid precautions regarding the SSU of the R8C/14.

Example of a source file:

```
.section test
mov.b      #10H, P1
mov.b      #03H, SSCRH
.end
```

Example of an output list file:

```
1                               .glb P1, SSCRH
2                               .section test
3   00000       C710E100        mov.b       #10H, P1
4   00004       C703B800        mov.b       #03H, SSCRH
                FE01            ; Generates code to escape precautions on the SSU or IIC register
5                               .end
```

Remarks: Use uppercase letters to specify this option.

Notes: For notes, see RENESAS TECHNICAL UPDATE.

When projects are built in the integrated development environment High-performance Embedded Workshop, open the Option menu and choose "Renesas M16C Standard Toolchain" and then "CPU" to set this option.

When projects are built in the integrated development environment TM, open the Option Browser and choose "CFLAGS" and then "General" or "AFLAGS" and then "Select Code Generation Target" to set this option.

This option cannot be specified simultaneously with the "-PATCH[6N]_TA" or "-PATCH[6N]_TAn" option.

There is no need to specify this option for the MCU groups that do not have the SSU or IIC function installed.

Section 6 Optimizing Linkage Editor Options

6.1 Option Specifications

6.1.1 Command Line Format

The format of the command line is as follows:

```
optlnk[{Δ<file name>|Δ<option string>}...]

        <option string>:-<option>[=<suboption>[,...]]
```

6.1.2 Subcommand File Format

The format of the subcommand file is as follows:

```
<option>{=|Δ}[<suboption>[,...]][Δ&][;<comment>]

        &: means line continuous.
```

For details, refer to section 5.2.8, Subcommand File Option.

6.2 List of Options

In the command line format in the following sections, uppercase letters indicate abbreviations. Underlined characters indicate the default settings.

The format of the corresponding dialog menus in the High-performance Embedded Workshop is as follows:

Tab name <Category>[Item]....

For details on dialog menus, refer to the High-performance Embedded Workshop.

The order of option description corresponds to that of the tabs and the categories in the High-performance Embedded Workshop.

6.2.1 Input Options

Table 6.1 Input Category Options

Item	**Command Line Format**	**Dialog Menu**	**Specification**
Input file	Input = <sub>[{,|Δ}…] <sub>: <file name> [(<module name>[,…])]	Link/Library <Input> [Show entries for :] [Relocatable files and object files]	Specifies input file. (Input file is specified without **input** on the command line.)
Library file	LIBrary = <file name>[,...]	Link/Library <Input> [Show entries for :] [Library files]	Specifies input library file.
Binary file	Binary = <sub> [,...] <sub>: <file name>(<section name> [:<boundary alignment>] [,<symbol name>])	Link/Library <Input> [Show entries for :] [Binary files]	Specifies input binary file.
Symbol definition	DEFine = <sub>[,…] <sub>: <symbol name> = {<symbol name> |<numerical value>}	Link/Library <Input> [Show entries for :] [Defines:]	Defines undefined symbols forcedly. Defined as the same value of symbol name. Defined as a numerical value.

Execution start address	ENTry = { <symbol name> |
	<address>}
		Link/Library <Input>
		[Use entry point :]
			Specifies an entry symbol.
			Specifies an entry address.

Prelinker	NOPRElink	Link/Library <Input> [Prelinker control :]	Disables prelinker initiation.

Input Input File

Link/Library <Input>[Show entries for :][Relocatable files and object files]

Format: Input = <suboption>[{, | Δ}…]

<suboption>: <file name>[(<module name>[,…])]

Description: Specifies an input file. Two or more files can be specified by separating them with a comma (,) or space.

Wildcards (* or ?) can also be used for the specification. String literals specified with wildcards are expanded in alphabetical order. Expansion of numerical values precedes that of alphabetical letters. Uppercase letters are expanded before lowercase letters.

Specifiable files are object files output from the compiler or the assembler, and relocatable or absolute files output from the optimizing linkage editor. A module in a library can be specified as an input file using the format of <library name>(<module name>). The module name is specified without an extension.

If an extension is omitted from the input file specification, **obj** is assumed when a module name is not specified and **lib** is assumed when a module name is specified.

Examples:

input=a.obj lib1(e)	; Inputs a.obj and module e in lib1.lib.
input=c*.obj	; Inputs all .obj files beginning with c.

Remarks: When **form=object** or **extract** is specified, this option is unavailable.

When an input file is specified on the command line, **input** should be omitted.

LIBrary Library File

Link/Library <Input>[Show entries for :][Library files]

Format: LIBrary = <file name>[,…]

Description: Specifies an input library file. Two or more files can be specified by separating them with a comma (,).

Wildcards (* or ?) can also be used for the specification. String literals specified with wildcards are expanded in the alphabetical order. Expansion of numerical values precedes that of alphabetical letters. Uppercase letters are expanded before lowercase letters.

If an extension is omitted from the input file specification, **lib** is assumed.

If **form=library** or **extract** is specified, the library file is input as the target library to be edited.

Otherwise, after the linkage processing between files specified for the input files are executed, undefined symbols are searched in the library file.

The symbol search in the library file is executed in the following order: user library files with the library option specification (in the specified order), the system library files with the library option specification (in the specified order), and then the default library (environment variable **HLNK_LIBRARY1,2,3**).

Examples:

library=a.lib,b	; Inputs a.lib and b.lib.
library=c*.lib	; Inputs all files beginning with **c** with the extension **.lib**.

Binary Binary File

Link/Library <Input>[Show entries for :][Binary files]

Format: Binary = <suboption>[,…]

<suboption>: <file name>(<section name>[:<boundary alignment>]
[,<symbol name>])

<boundary alignment>: 1 | 2 | 4 | 8 | 16 | 32 (default: 1)

Description: Specifies an input binary file. Two or more files can be specified by separating them with a comma (,).

If an extension is omitted for the file name specification, bin is assumed.

Input binary data is allocated as the specified section data. The section address is specified with the start option. That section cannot be omitted.

When a symbol is specified, the file can be linked as a defined symbol. For a variable name referenced by a C/C++ program, add an underscore (_) at the head of the reference name in the program.

The section specified with this option can have its section attribute and boundary alignment specified.

CODE or DATA can be specified for the section attribute.

If no section attributes are specified, the write, read, and execute attributes are all enabled by default.

A boundary alignment value can be specified for the section specified by this option. A power of 2 can be specified for the boundary alignment; no other values should be specified.

When the boundary alignment specification is omitted, 1 is used as the default.

Examples:

```
input=a.obj
start=P,D*/200
binary=b.bin(D1bin),c.bin(D2bin:4,_datab)
form=absolute
```

Allocates **b.bin** from 0x200 as the **D1bin** section.
Allocates **c.bin** after **D1bin** as the **D2bin** section (with boundary alignment = 4).
Links **c.bin** data as the defined symbol **_datab**.

Remarks: When **form={object | library}** or **strip** is specified, this option is unavailable.

If no input object file is specified, this option cannot be specified.

DEFine — Symbol Definition

Link/Library <Input>[Show entries for :][Defines]

Format: DEFine = <suboption>[,…]

<suboption>: <symbol name>={<symbol name> | <numerical value>}

Description: Defines an undefined symbol forcedly as an externally defined symbol or a numerical value.

The numerical value is specified in the hexadecimal notation. If the specified value starts with a letter from A to F, symbols are searched first, and if no corresponding symbol is found, the value is interpreted as a numerical value. Values starting with 0 are always interpreted as numerical values.

If the specified symbol name is a C/C++ variable name, add an underscore (_) at the head of the definition name in the program. If the symbol name is a C++ function name (except for the main function), enclose the definition name with the double-quotes including parameter strings. If the parameter is void, specify as "<function name>( )".

Examples:

```
define=_sym1=data  ; Defines _sym1 as the same value as
                   ; the externally defined symbol data.

define=_sym2=4000 ; Defines _sym2 as 0x4000.
```

Remarks: When **form={object | relocate | library}** is specified, this option is unavailable.

ENTry — Execution Start Address

Link/Library <Input>[Use entry point :]

Format: ENTry = {<symbol name> | <address>}

Description: Specifies the execution start address with an externally defined symbol or address.

The address is specified in hexadecimal notation. If the specified value starts with a letter from A to F, symbols are searched first, and if no corresponding symbol is found, the value is interpreted as an address. Values starting with 0 are always interpreted as addresses.

For a C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function name (except for the **main** function), enclose the definition name with double-quotes in the program including parameter strings. If the parameter is **void**, specify as "<function name>()".

If the **entry** symbol is specified at compilation or assembly, this option precedes the entry symbol.

Examples:

```
entry=_main        ; Specifies main function in C/C++ as the execution
                   ; start address.
entry="init()"     ; Specifies init function in C++ as the execution
                   ; start address.
entry=100          ; Specifies 0x100 as the execution start address.
```

Remarks: When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.

When optimization with undefined symbol deletion (**optimize=symbol_delete**) is specified, the execution start address should be specified. If it is not specified, the specification of the optimization with undefined symbol deletion is unavailable. When the CPU type is RX Family, optimization with undefined symbol deletion is not available when an address is specified with this option.

If the address is specified with this option, optimization for deleting unreferenced symbols is disabled.

NOPRElink — Prelinker

Link/Library <Input>[Show entries for :][Prelinker control :]

Format:	NOPRElink
Description:	Disables the prelinker initiation.
	The prelinker supports the functions to generate the C++ template instance automatically and to check types at run time. When the C++ template function and the runt-time type test function are not used, specify the **noprelink** option to reduce the link time.
Remarks:	This option is invalid when extract or strip is specified.
	If form=lib or form=rel is specified while the C++ template feature and runtime type information are in use, do not specify noprelink.

6.2.2 Output Options

Table 6.2 Output Category Options

Item	Command Line Format	Dialog Menu	Specification
Output format	FOrm ={ Absolute | Relocate | Object | Library [= {S|U}] | Hexadecimal | Stype | Binary }	Link/Library <Output> [Type of output file :]	Absolute format Relocatable format Object format Library format HEX format S-type format Binary format
Debugging information	DEBug SDebug NODEBug	Link/Library <Output> [Debug information :]	Output (in output file) Debugging information file output Not output
Record size unification	REcord={ H16 | H20 | H32 | S1 | S2 | S3 }	Link/Library <Output> [Data record header :]	HEX record Expansion HEX record 32-bit HEX record S1 record S2 record S3 record
ROM support function	ROm = <sub>[,…] <sub>:<ROM section name> =<RAM section name>	Link/Library <Output> [Show entries for :] [ROM to RAM mapped sections:]	Reserves an area in RAM for the relocation of a symbol with an address in RAM.
Output file	OUtput = <sub>[,…] <sub>:<file name> [=<output range>] <output range>: {<start address> -<end address> |<section name>[:…]}	Link/Library <Output> [Show entries for :] [Output file path/ Messages] or [Divided output files:]	Specifies output file (range specification and divided output are enabled)
External symbol-allocation information file	MAp [= <file name>]	Link/Library <Output> [Generate external symbol-allocation information file]	Specifies output of the external symbol-allocation information file (for SuperH Family and RX Family)
Output to unused area	SPace [= {<numerical value> | Random}]	Link/Library <Output> [Specify value filled in unused area] [Output padding data]	Specifies a value to output to unused area
Information message	Message NOMessage [= <sub>[,…]] <sub>:<error code> [-<error code>]	Link/Library <Output> [Show entries for :] [Output file path/ Messages] [Repressed information level messages:]	Output No output (error number specification and range specification are enabled)
Notification of unreferenced defined symbol	MSg_unused	Link/Library <Output> [Show entries for :] [Notify unused symbol:]	Notifies the user of the defined symbol which is never referenced
Reduce empty areas of boundary alignment	DAta_stuff	Link/Library <Output> [Show entries for :] [Reduce empty areas of boundary alignment:]	Reduces empty areas generated as the boundary alignment of sections after compilation (for SuperH Family and H8, H8S, H8SX Family)

Item	Command Line Format	Dialog Menu	Specification
Specification of data record byte count	BYte_count=<numerical value>	Link/Library <Output> [Length of data record :]	Specifies the maximum byte count of a data record
CRC	CRc = <suboption> <suboption>: <address where the result is output>=<target range> [/<polynomial expression>] [:<endian>] <address where the result is output>: <address> <target range>: <start address>-<end address>[,...] <polynomial expression>: { CCITT | 16 } <endian>: {BIG | LITTLE}	Link/Library <Output> [Show entries for :] [Generate CRC code]	Calculates the cyclic redundancy check (CRC) value for the target range at linkage and outputs the result to the specified address.
Filling padding data at section end	PADDING	Link/Library <Output> [Padding]	Outputs padding data to the end of a section to make the section match the boundary alignment.

Address setting for specified vector number	VECTN=<suboption>[,...]
	<suboption>:
	<vector number>={<symbol> | | <address>}
		Link/Library <Output> [Show entries for :] [Vector] [Specific vector :]	Assigns an address to the specified vector number in the variable vector table (for RX Family and M16C Series).

Address setting for unused variable vector area	VECT={<symbol>|<address>}	Link/Library <Output> [Show entries for :] [Vector] [Empty vector :]	Assigns an address to an unused area in the variable vector table (for RX Family and M16C Series).
utl30 information output	UTL	Link/Library <Output> [UTL information]	Outputs information for UTL30 (for M16C Series)
Jump table output	JUMP_ENTRIES_FOR_PIC=<section name>[…]	Link/Library <Output> [Jump table output]	Outputs a jump table (for RX Family)

FOrm — Output Format

Link/Library <Output>[Type of output file :]

Format: FOrm = {<u>Absolute</u> | Relocate | Object | Library[={S | <u>U</u>}]}
| Hexadecimal | Stype | Binary}

Description: Specifies the output format.

When this option is omitted, the default is **form=absolute**. Table 6.3 lists the suboptions.

form=relocate is not available when the RX Family CPU is selected.

Table 6.3 Suboptions of Form Option

Suboption	Description
absolute	Outputs an absolute file
relocate	Outputs a relocatable file
object	Outputs an object file. This is specified when a module is extracted as an object file from a library with the **extract** option.
library	Outputs a library file. When **library=s** is specified, a system library is output. When **library=u** is specified, a user library is output. Default is **library=u**.
hexadecimal	Outputs a **HEX** file. For details of the HEX format, refer to appendix 13.1.2, HEX File Format.
stype	Outputs an **S**-type file. For details of the **S**-type format, refer to appendix 13.1.1, S-Type File Format.
binary	Outputs a binary file.

Remarks: Table 6.4 shows relations between output formats and input files or other options.

Table 6.4 Relations Between Output Format And Input File Or Other Options

Output Format	Specified Option	Enabled File Format	Specifiable Option*1
Absolute	strip specified	Absolute file	input, output
	Other than above	Object file Relocatable file Binary file Library file	input, library, binary, debug/nodebug, sdebug, cpu, ps_check, start, rom, entry, output, map, hide, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, variable_forbid, function_forbid, section_forbid, absolute_forbid, profile, cachesize, sbr, compress, rename, delete, define, fsymbol, stack, noprelink, memory, msg_unused, data_stuff, show=symbol, reference, xreference
Relocate	extract specified	Library file	library, output
	Other than above	Object file Relocatable file Binary file Library file	input, library, debug/nodebug, output, hide, rename, delete, noprelink, msg_unused, data_stuff, show=symbol, xreference
Object	extract specified	Library file	Library, output
Hexadecimal Stype Binary		Object file Relocatable file Binary file Library file	input, library, binary, cpu, ps_check, start, rom, entry, output, map, space, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, variable_forbid, function_forbid, section_forbid, absolute_forbid, profile, cachesize, sbr, rename, delete, define, fsymbol, stack, noprelink, record, s9*2, byte_count*3, memory, msg_unused, data_stuff, show=symbol, reference, xreference
		Absolute file	input, output, record, s9*2, byte_count*3, show=symbol, reference, xreference
Library	strip specified	Library file	library, output, memory*4, show=symbol, section
	extract specified	Library file	library, output
	Other than above	Object file Relocatable file	input, library, output, hide, rename, delete, replace, noprelink, memory*4, show=symbol, section

Notes:

1. message/nomessage, change_message, logo/nologo, form, list, and subcommand can always be specified.
2. s9 can be used only when form=stype is specified for the output format.
3. byte_count can be used only when form=hexadecimal is specified for the output format.
4. memory cannot be used when hide is specified.

DEBug, SDebug, NODEBug Debugging Information

Link/Library <Output>[Debug information :]

Format:	<u>DEBug</u>
	SDebug
	NODEBug
Description:	Specifies whether debugging information is output.
	When **debug** is specified, debugging information is output to the output file.
	When **sdebug** is specified, debugging information is output to **<output file name>.dbg** file.
	When **nodebug** is specified, debugging information is not output.
	If **sdebug** and **form=relocate** are specified, **sdebug** is interpreted as **debug**.
	If **debug** is specified and if two or more files are specified to be output with **output**, they are interpreted as **sdebug** and debugging information is output to **<first output file name>.dbg**.
	When this option is omitted, the default is **debug**.
Remarks:	When **form={object | library | hexadecimal | stype | binary}**, **strip** or **extract** is specified, this option is unavailable.

REcord Record Size Unification

Link/Library <Output>[Data record header :]

Format:	REcord = { H16 | H20 | H32 | S1 | S2 | S3 }
Description:	Outputs data with the specified data record regardless of the address range.
	If there is an address that is larger than the specified data record, the appropriate data record is selected for the address.
	When this option is omitted, various data records are output according to each address.
Remarks:	This option is available only when **form=hexadecimal** or **stype** is specified.

ROm — ROM Support Function

Link/Library <Output>[Show entries for :][ROM to RAM mapped sections]

Format: ROm = <suboption>[,…]

<suboption>: <ROM section name>=<RAM section name>

Description: Reserves ROM and RAM areas in the initialized data area and relocates a defined symbol in the ROM section with the specified address in the RAM section.

Specifies a relocatable section including the initial value for the ROM section.

Specifies a nonexistent section or relocatable section whose size is 0 for the RAM section.

Examples: rom=D=R
start=D/100,R/8000

Reserves **R** section with the same size as **D** section and relocates defined symbols in **D** section with the **R** section addresses.

Remarks: When **form={object | relocate | library}**or **strip** is specified, this option is unavailable.

OUtput — Output File

Link/Library <Output> [Show entries for :][Output file path/ Messages] or [Divided output files]

Format: OUtput = <suboption>[,…]

<suboption>: <file name>[=<output range>]

<output range>: {<start address>-<end address> | <section name>[:…]}

Description: Specifies an output file name. When **form=absolute**, **hexadecimal**, **stype**, or **binary** is specified, two or more files can be specified. An address is specified in the hexadecimal notation. If the specified data starts with a letter from A to F, sections are searched first, and if no corresponding section is found, the data is interpreted as an address. Data starting with 0 are always interpreted as addresses.

When this option is omitted, the default is <first input file name>.<default extension>.

The default extensions are as follows:

form=absolute: abs	form=relocate: rel	form=object: obj
form=library: lib	form=hexadecimal: hex	form=stype: mot
form=binary: bin		

Examples: output=file1.abs=0-ffff,file2.abs=10000-1ffff

Outputs the range from 0 to 0xffff to **file1.abs** and the range from 0x10000 to 0x1ffff to **file2.abs**.

output=file1.abs=sec1:sec2,file2.abs=sec3

Outputs the **sec1** and **sec2** sections to **file1.abs** and the **sec3** section to **file2.abs**.

Remarks: When a file is output in section units while the CPU type is RX Family in big endian, the section size should be a multiple of 4.

MAp — Output of External Symbol Allocation Information File

Link/Library <Output>[Generate external symbol-allocation information file]

Format: MAp [= <file name>]

Description: Outputs the external-symbol-allocation information file that is used by the compiler in optimizing access to external variables.

When <file name> is not specified, the file has the name specified by the **output** option or the name of the first input file, and the extension **bls**.

If the order of the declaration of variables in the external-symbol-allocation information file is not the same as the order of the declaration of variables found when the object was read after compilations, an error will be output.

Remarks: This option is valid only when **form={absolute | hexadecimal | stype | binary}** is specified.

This option is available when the CPU type is SuperH Family or RX Family.

SPace — Output to Unused Areas

Link/Library <Output>[Show entries for :][Specify value filled in unused area]
[Output padding data]

Format: SPace [= {<numerical value> | Random}]

Description: Fills the unused areas in the output ranges with random values or a user-specified hexadecimal value.

The following unused areas are filled with the value according to the output range specification in the **output** option:

When section names are specified for the output range:
The specified value is output to unused areas between the specified sections.

When an address range is specified for the output range:
The specified value is output to unused areas within the specified address range.

A 1-, 2-, or 4-byte value can be specified. The number of hexadecimal digits specified to the **space** option determines the size of the <numerical value>. If a 3-byte value is specified, the upper digit is extended with 0 to use it as a 4-byte value. If an odd number of digits are specified, the upper digits are extended with 0 to use it as an even number of digits.

If the size of an unused area is not a multiple of the size of the specified value, the value is output as many times as possible, then a warning message is output.

Remarks: When no numerical value is specified by this option, unused areas are not filled with values.

This option is available only when **form={binary | stype | hexadecimal}** is specified.

When no output range is specified by the **output** option, this option is unavailable.

Message, NOMessage

Information Message

Link/Library <Output>[Show entries for :] [Output file path/ Messages]
[Repressed information level messages :]

Format: Message

NOMessage [=<suboption>[,…]]

<suboption>: <error number>[-<error number>]

Description: Specifies whether information level messages are output.

When **message** is specified, information level messages are output.

When **nomessage** is specified, the output of information level messages are disabled. If an error number is specified, the output of the error message with the specified error number is disabled. A range of error message numbers to be disabled can be specified using a hyphen (-). If a warning or error level message number is specified, the message output is disabled assuming that **change_message** has changed the specified message to the information level.

When this option is omitted, the default is **nomessage**.

Examples: nomessage=4,200-203,1300

Messages of L0004, L0200 to L0203, and L1300 are disabled to be output.

MSg_unused Notification of Unreferenced Symbol

Link/Library <Output>[Show entries for :] [Output Messages] [Notify unused symbol:]

Format:	MSg_unused
Description:	Notifies the user of the externally defined symbol which is not referenced during linkage through an output message.
Examples:	optlnk -msg_unused a.obj
Remarks:	When an absolute file is input, this option is invalid.

To output a notification message, the **message** option must also be specified.

The linkage editor may output a message for the function that was inline-expanded at compilation. To avoid this, add a **static** declaration for the function definition.

In any of the following cases, references are not correctly analyzed so that information shown by output messages will be incorrect.

- **goptimize** is not specified at assembly and there are branches to the same section within the same file (only when an H8, H8S, H8SX Family CPU is specified).
- There are references to constant symbols within the same file.
- There are branches to immediate subordinate functions when optimization is specified at compilation.
- The external variable access optimization is valid at compilation (only when an SuperH Family CPU is specified).
- An offset value is directly specified in a **#pragma tbr** in the C source program (only when the SH-2A or SH2A-FPU is specified as the CPU).
- Optimization is specified at linkage and constants or literals are unified.

DAta_stuff — Reduce empty areas of boundary alignment

Link/Library <Output>[Show entries for :] [Reduce empty areas of boundary alignment:]

Format: DAta_stuff

Description: At linkage, reduces empty areas of boundary alignment. This option affects constant, initialized and uninitialized data areas.

When this option is specified, empty areas generated as the boundary alignment of sections after compilation are filled at linkage. However, the order of data allocation is not changed.

When this option is not specified, linkage is based on the boundary alignment of sections after compilation.

Specifying this option fills the unnecessary empty areas generated by boundary alignment, reducing the size of the data sections as a whole.

Examples:

```
<tp1.c>              <tp2.c>
long a;              char d;
char b,c;            long e;
                     char f;
```

Sizes of data sections after compilation (taking the output of the SuperH Family compiler as an example):

tp1.obj: 4 + 1 + 1 = 6 bytes
tp2.obj: 1 + 3 [*] + 4 + 1 = 9 bytes

Sizes of data sections for **tp1.obj** and **tp2.obj** after linkage:

1) When **data_stuff** is not specified
Object files are linked based on the boundary alignment of the sections (conventional process).
6 bytes [tp1] + 2 bytes [*] + 9 bytes [tp2] = 17 bytes

2) When **data_stuff** is specified
Linkage is performed with filling of the unnecessary empty spaces generated between sections by boundary alignment.
(4 + 1 + 1) bytes + 1 byte + 1 byte [*] + 4 bytes + 1 byte = 13 bytes

Notes:

1. * indicates an empty area generated by boundary alignment.
2. The sizes of the data sections after compilation may differ from those in the above example according to the specification of

Remarks: Correct operation is not guaranteed if this option is specified when an object file compiled with the smap option of the SuperH Family compiler is linked.

The function of this option is not applicable to object files generated by the assembler.

Specification of this option is invalid in any of the following cases:

- **form=library, object or relocate** is specified
- An absolute load module is input
- **memory=low** is specified
- **nooptimize** is not specified

Optimization will not be applied in the linkage of a relocatable file that was generated with this option specified.

This option is unavailable when the CPU type is RX Family, M16C Series, or R8C Family.

BYte_count — Specification of Data Record Byte Count

Link/Library <Output>[Length of data record :]

Format:	BYte_count=<numerical value>
Description:	Specifies the maximum byte count for a data record when a file is to be created in the **Intel-Hex** format. Specify a one-byte hexadecimal value (01 to FF) for the byte count. When this option is not specified, the linkage editor assumes FF as the maximum byte count when creating an **Intel-Hex** file.
Examples:	byte_count=10
Remarks:	This option is invalid when the file to be created is not an **Intel-Hex**-type (**form=hex**) file.

CRc — CRC

Link/Library <Output> [Show entries for :] [Generate CRC code]

Format: CRc = <suboption>

<suboption>: <address where the result is output>=<target range>
[/<polynomial expression>][:<endian>]

<address where the result is output>: <address>

<target range>: <start address>-<end address>[,...]

<polynomial expression>: { CCITT | 16 }

<endian>: {BIG | LITTLE}

Description: This option is used for cyclic redundancy checking (CRC) of values from the lowest to the highest address of each target range and outputs the calculation result to the specified address.

<endian> can be specified only when the CPU type is RX Family. When <endian> is specified, the calculation result is output to the specified address in the specified endian. When <endian> is not specified, the result is output to the specified address in the endian used in the absolute file.

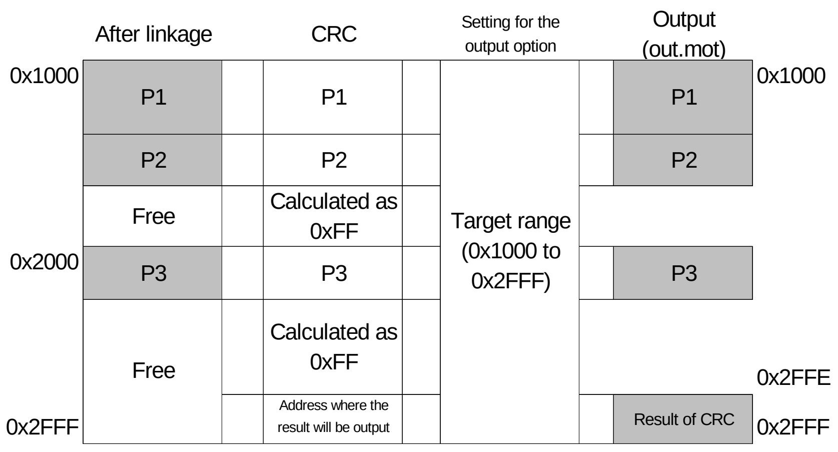
CRC-CCITT or **CRC-16** is selectable as a polynomial expression (default: **CRC-CCITT**).

Polynomial expression:

CRC-CCITT
X^16+X^12+X^5+1
In bit expression: (10001000000100001)

CRC-16
X^16+X^15+X^2+1
In bit expression: (11000000000000101)

Example 1: optlnk *.obj -form=stype -start=P1,P2/1000,P3/2000
-crc=2FFE=1000-2FFD -output=out.mot=1000-2FFF



crc option: -crc=2FFE=1000-2FFD
In this example, CRC will be calculated for the range from 0x1000 to 0x2FFD and the result will be output to address 0x2FFE.
When the **space** option has not been specified, **space=0xFF** is assumed for calculation of free areas within the target range.

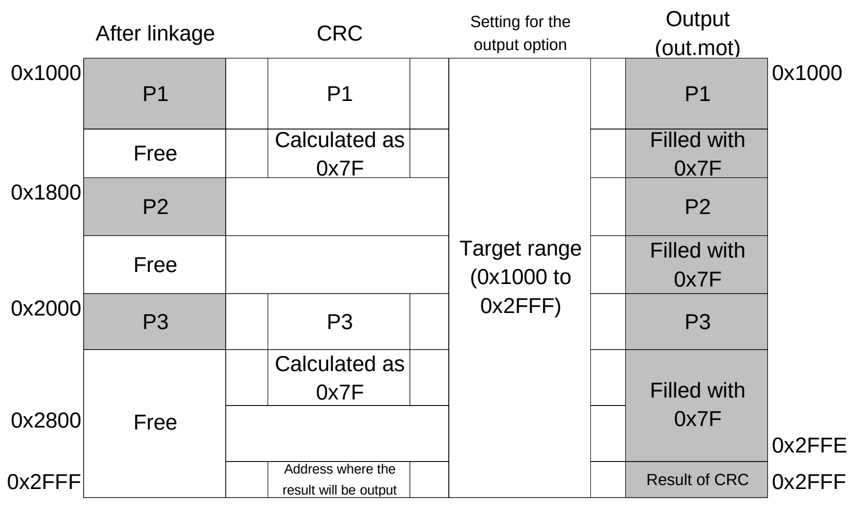
output option: -output=out.mot=1000-2FFF
Since the space option has not been specified, the free areas are not output to the out.mot file. 0xFF is used in CRC for calculation of the free areas, but will not be filled into these areas.

Notes:

1. The address where the result of CRC will be output cannot be included in the target range.
2. The address where the result of CRC will be output must be included in the output range specified with the output option.

Example 2:

```
optlnk *.obj -form=stype -start=P1/1000,P2/1800,P3/2000
          -space=7F -crc=2FFE=1000-17FF,2000-27FF
          -output=out.mot=1000-2FFF
```



crc option: -crc=2FFE=1000-17FF,2000-27FF
In this example, CRC will be calculated for the two ranges, 0x1000 to 0x17FF and 0x2000 to 0x27FF, and the result will be output to address 0x2FFE.
Two or more non-contiguous address ranges can be selected as the target range for CRC.

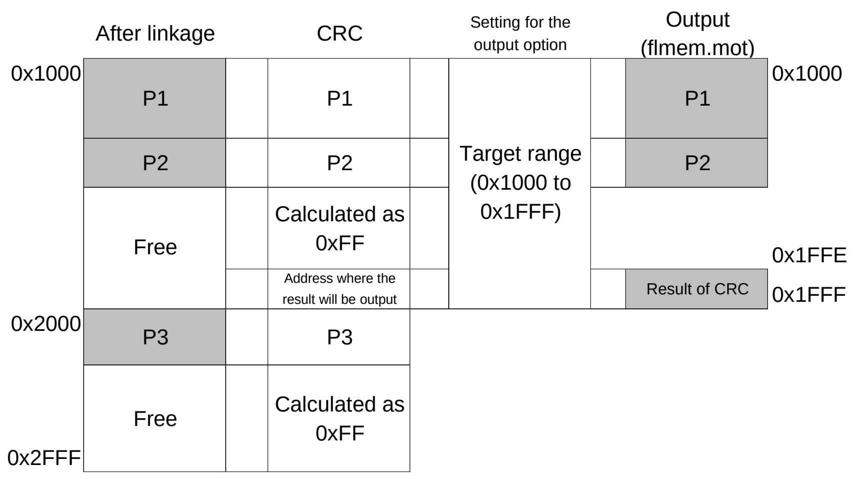
space option: -space=7F
The value of the **space** option (0x7F) is used for CRC in free areas within the target range.

output option: -output=out.mot=1000-2FFF
Since the **space** option has been specified, the free areas are output to the **out.mot** file. 0x7F will be filled into the free areas.

Notes:

1. The order that CRC is calculated for the specified address ranges is not the order that the ranges have been specified. CRC proceeds from the lowest to the highest address.
2. Even if you wish to use the crc and space options at the same time, the space option cannot be set as random or a value of 2 bytes or more. Only 1-byte values are valid.

Example 3: optlnk *.obj -form=stype -start=P1,P2/1000,P3/2000
-crc=1FFE=1000-1FFD,2000-2FFF
-output=flmem.mot=1000-1FFF



crc option: -crc=1FFE=1000-1FFD,2000-2FFF
In this example, CRC will be calculated for the two ranges, 0x1000 to 0x1FFD and 0x2000 to 0x2FFF, and the result will be output to address 0x1FFE.
When the **space** option has not been specified, **space=0xFF** is assumed for calculation of free areas within the target range.

output option: -output=flmem1.mot=1000-1FFF
Since the **space** option has not been specified, the free areas are not output to the **flmem1.mot** file. 0xFF is used in CRC for calculation of the free areas, but will not be filled into these areas.

Remarks: This option is invalid when two or more absolute files have been selected.

This option is valid only when **form={hexadecimal | stype}**.

When the **space** option has not been specified and the target range includes free areas that will not be output, the linkage editor assumes in CRC that 0xFF has been set in the free areas.

An error occurs if the target range includes an overlay area.

Sample Code:

The sample code shown below is provided to check the result of CRC figured out by the crc option. The sample code program should match the result of CRC by optlnk.

When the selected polynomial expression is **CRC-CCITT**:

```
typedef  unsigned       char    uint8_t;
typedef  unsigned       short   uint16_t;
typedef  unsigned       long    uint32_t;

uint16_t CRC_CCITT(uint8_t *pData, uint32_t iSize)
{
   uint32_t   ui32_i;
   uint8_t    *pui8_Data;
   uint16_t   ui16_CRC = 0xFFFFu;

   pui8_Data = (uint8_t *)pData;

   for(ui32_i = 0; ui32_i < iSize; ui32_i++)
   {
       ui16_CRC  = (uint16_t)((ui16_CRC >> 8u) |
                           ((uint16_t)((uint32_t)ui16_CRC << 8u)));
       ui16_CRC ^= pui8_Data[ui32_i];
       ui16_CRC ^= (uint16_t)((ui16_CRC & 0xFFu) >> 4u);
       ui16_CRC ^= (uint16_t) ((ui16_CRC << 8u) << 4u);
       ui16_CRC ^= (uint16_t)(((ui16_CRC & 0xFFu) << 4u) << 1u);
   }
ui16_CRC = (uint16_t)( 0x0000FFFFul &
                 ((uint32_t)~(uint32_t)ui16_CRC) );
   return ui16_CRC;
}
```

When the selected polynomial expression is **CRC-16**:

```
#define POLYNOMIAL 0xa001 // Generated polynomial expression CRC-16

typedef  unsigned       char    uint8_t;
typedef  unsigned       short   uint16_t;
typedef  unsigned       long    uint32_t;

uint16_t CRC16(uint8_t *pData, uint32_t iSize)
{
   uint16_t crcdData = (uint16_t)0;
   uint32_t data = 0;
   uint32_t i,cycLoop;

   for(i=0;i<iSize;i++){
       data = (uint32_t)pData[i];
       crcdData = crcdData ^ data;
       for (cycLoop = 0; cycLoop < 8; cycLoop++) {
           if (crcdData & 1) {
               crcdData = (crcdData >> 1) ^ POLYNOMIAL;
           } else {
               crcdData = crcdData >> 1;
           }
       }
   }
   return crcdData;
}
```

PADDING — Filling padding data at section end

Format:	PADDING
Description:	Fills in padding data at the end of a section so that the section size is a multiple of the boundary alignment of the section.
Examples:	-start=P,C/0 –padding When the boundary alignment of section **P** is 4 bytes, the size of section **P** is 0x06 bytes, the boundary alignment of section **C** is 1 byte, and the size of section **C** is 0x03 bytes, two bytes of padding data is filled in section **P** to make its size become 0x08 bytes and then linkage is performed. -start=P/0,C/7 –padding When the boundary alignment of section **P** is 4 bytes, the size of section **P** is 0x06 bytes, the boundary alignment of section **C** is 1 byte, and the size of section **C** is 0x03 bytes, if two bytes of padding data is filled in section **P** to make its size become 0x08 bytes and then linkage is performed, error L2321 will be output because section **P** overlaps with section **C**.
Remarks:	The value of the created padding data is 0x00. Since padding is not performed to an absolute address section, the size of an absolute address section should be adjusted by the user. This option is valid when the CPU type is SuperH Family or RX Family.

VECTN — Address Setting for Specified Vector Number

Link/Library <Output> [Show entries for:] [Address allocation on specific vector]

Format: VECTN = <suboption>[,...]

<suboption>: <vector number> = {<symbol> | <address>}

Description: Assigns the specified address to the specified vector number in the variable vector table (**C$VECT** section).

When this option is specified, a variable vector table is created as a **C$VECT** section and the specified address is set in the table even if there is no interrupt function in the source code.

Specify a decimal value from 0 to 255 for <vector number>.

Specify the external name of the target function for <symbol>.

Specify the desired hexadecimal address for <address>.

Examples:

```
-vectn=30=_f1,31=0000F100 ; Specifies the _f1 address for vector
                          ; number 30 and 0x0f100 for vector
                          ; number 31
```

Remarks: This option is valid when the CPU type is RX Family, M16C Series, or R8C Family.

This option is ignored when the user creates a **C$VECT** section in the source program because the variable vector table is not automatically created in this case.

VECT — Address Setting for Unused Vector Area

Link/Library <Output> [Show entries for:] [Filling address on empty vector]

Format: VECT={<symbol>|<address>}

Description: Assigns the specified address to the vector number to which no address has been assigned in the variable vector table (**C$VECT** section).

When this option is specified, a variable vector table is created as a **C$VECT** section by the linkage editor and the specified address is set in the table even if there is no interrupt function in the source code.

Specify the external name of the target function for <symbol>.

Specify the desired hexadecimal address for <address>.

Remarks: This option is valid when the CPU type is RX Family, M16C Series, or R8C Family.

This option is ignored when the user creates a **C$VECT** section in the source program because the variable vector table is not automatically created in this case.

When the **{<symbol>|<address>}** specification is started with 0, the whole specification is assumed as an address.

UTL

utl30 information output

Link/Library <Other> [Other option] [utl file output]

Format: UTL

Description: Generates an external file (utl file) used for input to the tool (utl30) included with the compiler package.

The generated file is assigned a name “<output file name>.utl.”

Examples:

```
tp.obj
utl
output=test.abs
```

Outputs inspector information from **tp.obj** to **test.utl**.

Remarks: This option is valid only when the compiler for the M16C microcontrollers is used.

This option cannot be used when processing the abs files input to the linkage editor.

This option is invalid when form={object | library} is specified.

JUMP_ENTRIES_FOR_PIC

Jump table output

Link/Library <Output> [Jump table]

Format: JUMP_ENTRIES_FOR_PIC=<section name>[,…]

Description: Outputs an assembler source for a jump table to branch into external definition symbols in the specified section.

The file name is <output file>.jmp.

Example:

```
jump_entries_for_pic=sct2,sct3
output=test.abs
```

A jump table for branching into external definition symbols in the sections **sect2** and **sect3** is output to **test.jmp**.

[Example of a file output to **test.jmp**]

```
        .glb _func01
        .glb _func02
        .SECTION P,CODE
_func01:
        MOV.L #1000H,R14
        JMP     R14
_func02:
        MOV.L #2000H,R14
        JMP     R14
        .END
```

Remarks: This option is invalid when **form={object | relocate| library}** or strip is specified.

This option is invalid when the CPU type is other than the RX series.

The generated jump table is output to the P section.

Types of sections specifiable for the section name are only the program section.

6.2.3 List Options

Table 6.5 List Category Options

Item	Command Line Format	Dialog Menu	Specification
List file	LISt [= <file name>]	Link/Library <List> [Generate list file]	Specifies the output of list file.
List contents	SHow [= <sub>[,...]] <sub>: {SYmbol | Reference | SEction | Xreference | Total_size | VECTOR | ALL }	Link/Library <List> [Contents :]	 Symbol information Number of references Section information Cross-reference information Total sizes of sections Vector Information All information

LISt List File

Link/Library <List> [Generate list file]

Format: LISt [=<file name>]

Description: Specifies list file output and a list file name.

If no list file name is specified, a list file with the same name as the output file (or first output file) is created, with the extension **lbp** when **form=library** or **extract** is specified, or **map** in other cases.

SHow — List Contents

Link/Library <List> [Contents]

Format: SHow [=<sub>[,…]]
<sub>:{ SYmbol | Reference | SEction | Xreference | Total_size | VECTOR | ALL}

Description: Specifies output contents of a list.

Table 6.6 lists the suboptions.

For details of list examples, refer to section 7.3, Linkage List, and section 7.4, Library List in the user's manual.

Table 6.6 Suboptions of show Option

Output Format	Suboption Name	Description
form=library or **extract** is specified.	symbol	Outputs a symbol name list in a module
	reference	Not specifiable
	section	Outputs a section list in a module
	xreference	Not specifiable
	total_size	Not specifiable
	vector	Not specifiable
	all	Not specifiable (when **extract** is specified) Outputs a symbol name list and a section list in a module (when **form=library**)
Other than **form=library** and **extract** is not specified.	symbol	Outputs symbol address, size, type, and optimization contents.
	reference	Outputs the number of symbol references.
	section	Not specifiable
	xreference	Outputs the cross-reference information.
	total_size	Shows the total sizes of sections allocated to the ROM and RAM areas.
	vector	Outputs vector information.
	all	If **form=rel**, the linkage editor outputs the same information as when **show=symbol**, **xreference**, or **total_size** is specified. If **form=rel** and **data_stuff** have been specified, the linkage editor outputs the same information as when **show=symbol** or **total_size** is specified. If **form=abs**, the linkage editor outputs the same information as when **show=symbol**, **reference**, **xreference**, or **total_size** is specified. If **form=hex**, **stype**, or **bin**, the linkage editor outputs the same information as when **show=symbol**, **reference**, **xreference**, or **total_size** is specified. If **form=obj**, **all** is not specifiable.

Remarks: The following table shows whether suboptions will be valid or invalid by all possible combinations of options **form**, **show**, and/or **show=all**.

		Symbol	**Reference**	**Section**	**Xreference**	**Vector**	**Total_size**
form=abs	show	Valid	Valid	Invalid	Invalid	Invalid	Invalid
	show=all	Valid	Valid	Invalid	Valid	Valid	Valid
form=lib	show	Valid	Invalid	Valid	Invalid	Invalid	Invalid
	show=all	Valid	Invalid	Valid	Invalid	Invalid	Invalid
form=rel	show	Valid	Invalid	Invalid	Invalid	Invalid	Invalid
	show=all	Valid	Invalid	Invalid	Valid*	Invalid	Valid
form=obj	show	Valid	Valid	Invalid	Invalid	Invalid	Invalid
	show=all	Valid	Invalid	Invalid	Invalid	Invalid	Invalid
form=hex/bin/sty	show	Valid	Valid	Invalid	Invalid	Invalid	Invalid
	show=all	Valid	Valid	Invalid	Valid	Valid*	Valid*

Note: The option is invalid if an absolute-format file is input.

Note the following limitations on output of the cross-reference information.

- When the relocatable format is specified for the output file and the **data_stuff** option is specified, no cross-reference information is output.
- When an absolute-format file is input, the referrer address information is not output.
- When **-goptimize** is not specified at assembly, information about branches to the same section within the same file is not output (only when an H8, H8S, H8SX Family CPU is specified).
- Information about references to constant symbols within the same file is not output.
- When optimization is specified at compilation, information about branches to immediate subordinate functions is not output.
- When optimization of access to external variables is specified, information about references to variables other than base symbols is not output (only when an SuperH Family or RX Family CPU is specified).
- When an offset value is directly specified in a **#pragma tbr** in the C source program, information about that function is not output (only when the SH-2A or SH2A-FPU is specified as the CPU).
- When optimization is specified at linkage and constants or literals are unified, information about references to these constants or literals is not output.
- Both **show=total_size** and **total_size** output the same information.
- **show=vector** can be used when the CPU type is RX Family, M16C Series, or R8C Family.
- When **show=reference** is valid, the number of references of the variable specified by **#pragma address** is output as 0 (only when a SuperH Family or RX Family CPU is specified).

6.2.4 Optimize Options

Table 6.7 Optimize Category Options

Item	Command Line Format	Dialog Menu	Specification
Optimization	OPtimize = <sub>[…] <sub>: {STring_unify | SYmbol_delete | Variable_access | Register | SAMe_code | SHort_format | Function_call | Branch | Speed | SAFe } NOOPtimize}	Link/Library <Optimize> [Show entries for :] [Optimize items] [Optimize :]	Executes optimization. Unifies constants/string literals. Deletes unreferenced symbols. Uses short absolute addressing mode. Provides optimization with register save/restore. Unifies same codes. Shortens the addressing mode. Uses indirect addressing mode. Provides optimization for branches. Provides optimization for speed. Provides safe optimization. No optimization.
Same code size	SAMESize = <size> (default: sames=1e)	Link/Library <Optimize> [Eliminated size :]	Specifies the minimum size to unify same codes.
Profile information	PROfile = <file name>	Link/Library <Optimize> [Include profile :]	Specifies a profile information file. (Dynamic optimization is provided.)

Cache size	CAchesize=<sub>
	<sub>: Size=<size> |
	Align=<line size>
	(default: ca=s=8,a=20)
		Link/Library <Optimize>
		[Cache size :]
			Specifies a cache size.
			Specifies a cache line size.
			(for SuperH Family)

Optimization partially disabled	SYmbol_forbid= <symbol name>[,…] SAMECode_forbid= <function name>[,…] Variable_forbid= <symbol name>[,…] FUnction_forbid= <function name>[,…] SEction_forbid = <sub>[,...] <sub>: [<file name>| <module name>] (<section name>[,...]) Absolute_forbid= <address>[+<size>][,…]	Link/Library <Optimize> [Show entries for :] [Forbid item]	Specifies a symbol where unreferenced symbol deletion is disabled. Specifies a symbol where same code unification is disabled. Specifies a symbol where short absolute addressing mode is disabled. Specifies a symbol where indirect addressing mode is disabled. Specifies a section where optimization is disabled. Specifies an address range where optimization is disabled.

OPtimize, NOOPtimize

Optimization

Link/Library <Optimize> [Show entries for :][Optimize items][Optimize :]

Format: OPtimize [= <suboption>[,…]]

NOOPtimize

<suboption>: { STring_unify | SYmbol_delete | Variable_access | Register
| SAMe_code | SHort_format | Function_call | Branch | SPeed
| SAFe }

Description: Specifies whether the inter-module optimization is executed.

When **optimize** is specified, optimization is performed for the file specified with the **goptimize** option at compilation or assembly.

When **nooptimize** is specified, no optimization is executed for a module.

When this option is omitted, the default is **optimize**.

Table 6.8 shows the suboptions

Table 6.8 Suboptions of Optimize Option

Suboption	Description	Program to be Optimized*1							
		SHC	SHA	H8C	H8A	RXC	RXA	NCC	NCA
No parameter	Provides all optimizations	O	×	O	O	O	O	O	×
string_unify	Unifies same-value constants having the **const** attribute. Constants having the **const** attribute are: ● Variables defined as const in C/C++ program ● Initial value of character string data ● Literal constant	O	×	O	×	×	×	×	×
symbol_delete	Deletes variables/functions that are not referenced. Always be sure to specify **#pragma** entry at compile time or the **entry** option in optlnk.	O	×	O	×	O	×	×	×
variable_access	Allocates frequently accessed variables to the area accessible in the 8/16 bit absolute addressing mode. The **cpu** option should be specified at compilation and assembly.	×	×	O	O	×	×	×	×

Suboption	Description	Program to be Optimized*1							
		SHC	SHA	H8C	H8A	RXC	RXA	NCC	NCA
register	Investigates function calls, relocates registers and deletes redundant register save or restore codes. Always be sure to specify **#pragma** entry at compile time or the **entry** option in optlnk.	O	×	O	×	×	×	×	×
same_code	Creates a subroutine for the same instruction sequence.	O	×	O	×	×	×	×	×
short_format	Replaces an instruction having a displacement or an immediate value with a smaller-size instruction when the code size of the displacement or immediate value can be reduced.	×	×	O	O	×	×	×	×
function_call	Allocates addresses of frequently accessed functions to the range 0 to 0xFF if there is a space. When the CPU is H8SX Family, the following ranges are also used: H8SXN: 0x100 to 0x1FF H8SXM,H8SXA,H8SXX: 0x200 to 0x3FF The **cpu** option should be specified at compilation and assembly.	×	×	O	O	×	×	×	×
branch	Optimizes branch instruction size according to program allocation information. Even if this option is not specified, it is performed when any other optimization is executed.	O	×	O	O	O	O	O	×
speed	Executes optimizations other than those reducing object speed. This suboption is the same as the following specifications: optimize=string_unify, symbol_delete, variable_access, register, short_format, or branch	O	×	O	O	O*2	×	O*2	×
safe	Executes optimizations other than those limited by variable or function attributes. This suboption is the same as the following specifications: optimize=string_unify, register, short_format, or branch	O	×	O	O	O*4	×	O*3	×

Notes: 1. SHC: C/C++ program for SuperH Family
SHA: Assembly program for SuperH Family
H8C: C/C++ program for H8, H8S, H8SX Family
H8A: Assembly program for H8, H8S, H8SX Family
RXC: C/C++ program for RX Family,
RXA: Assembly program for RX Family
NCC: C/C++ program for M16C Series, or R8C Family
NCA: Assembly program for M16C Series, or R8C Family
2. **symbol_delete**, **branch**, and **short_format** are valid.
3. **branch** is valid.
4. **short_format** and **branch** are valid

Remarks: When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.
When optimization of access to external variables is specified at compilation, optimization with unification of constants/string literals (**optimize=string_unify**) is invalid.
optimize=short_format is available only when the CPU is H8SX Family.
When the CPU is SH-2A or SH2A-FPU, the code size may increase due to the **optimize=register** function.
When a start function with **#pragma entry** or **entry** is not specified, **optimize=symbol_delete** is invalid.

SAMesize — Common Code Size

Link/Library <Optimize> [Eliminated size :]

Format: SAMESize = <size>

Description: Specifies the minimum code size for the optimization with the same-code unification (**optimize=same_code**). Specify a hexadecimal value from 8 to 7FFF.

When this option is omitted, the default is **samesize=1E**.

Remarks: When **optimize=same_code** is not specified, this option is unavailable.

PROfile — Profile Information

Link/Library <Optimize> [Include profile :]

Format: PROfile = <file name>

Description: Specifies a profile information file.

Specifiable profile information files are those output from the High-performance Embedded Workshop Ver. 2.0 or later.

When a profile information file is specified, inter-module optimization according to dynamic information can be performed.

Table 6.9 shows optimizations influenced by a profile information input.

Table 6.9 Relations Between Profile Information and Optimization

Suboption	Description	Program to be Optimized*1			
		SHC	SHA	H8C	H8A
variable_access	Allocates variables from those that are dynamically accessed more frequently.	×	×	O	O
function_call	Lowers the optimizing priority of functions that are dynamically accessed frequently.	×	×	O	O
branch	Allocates a function that is dynamically accessed frequently near the calling function. For the SH program, the optimization with allocation is performed depending on the cache size specified using the **cachesize** option.	O	Δ*2	O	Δ

Notes: 1. SHC: C/C++ program for SuperH Family
SHA: Assembly program for SuperH Family
H8C: C/C++ program for H8, H8S, H8SX Family
H8A: Assembly program for H8, H8S, H8SX Family
2. Movement is provided not in the function unit, but in the input file unit.

Remarks: When the **optimize** option is not specified, this option is unavailable.

CAchesize — Cache Size

Link/Library <Optimize> [Cache size :]

Format:	CAchesize = <suboption>
	<suboption>: Size = <size> | Align = <line size>
Description:	Specifies a cache size and cache line size.
	When **profile** is specified, this option is used at the branch instruction optimization (**optimize=branch**).
	Specify the size in Kbytes and specify the line size in bytes in the hexadecimal notation.
	When this option is omitted, the default is **cachesize=size=8, align=20**.
Remarks:	If **profile** is not specified, this option is unavailable.

SYmbol_forbid, SAMECode_forbid, Variable_forbid, FUnction_forbid, SEction_forbid, Absolute_forbid — Optimization Partially Disabled

Link/Library <Optimize> [Show entries for :] [Forbid item]

Format:	SYmbol_forbid = <symbol name> [,…]
	SAMECode_forbid = <function name> [,…]
	Variable_forbid = <symbol name> [,…]
	FUnction_forbid = <function name> [,…]
	SEction_forbid = <sub>[,…]
	<sub>: [<file name>|<module name>](<section name>[,…])
	Absolute_forbid = <address> [+<size>] [,…]
Description:	Disables optimization for the specified symbol, section, or address range. Specify an address or the size in the hexadecimal notation. For a C/C++ variable or C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function, enclose the definition name in the program with double-quotes including the parameter strings. When the parameter is **void**, specify as "<function name>( )".
	Table 6.10 shows the suboptions.

Table 6.10 Suboptions of Optimization Partially Disabling Option

Suboption	Parameter	Description
symbol_forbid	Function name | variable name	Disables optimization regarding unreferenced symbol deletion
samecode_forbid	Function name	Disables optimization regarding same-code unification
variable_forbid	Variable name	Disables optimization regarding short absolute addressing mode
function_forbid	Function name	Disables optimization regarding indirect addressing mode
section_forbid	Section name File name Module name	Disables optimization for the specified section. If an input file name or library module name is also specified, the optimization can be disabled for a specific file, not only the entire section.
absolute_forbid	Address [+ size]	Disables optimization regarding address + size specification

Examples:

symbol_forbid="f(int)" ; Does not delete the C++ function **f(int)**
; even if it is not referenced.

section_forbid=(P1) ; Disables any optimization for section
; **P1**.

section_forbid=a.obj(P1,P2) ; Disables any optimization for sections
; **P1** and **P2** in **a.obj**.

Remarks: If optimization is not applied at linkage, this option is ignored.

To disable optimization for an input file with its path name, type the path with the file name when specifying **section_forbid**.

6.2.5 Section Options

Table 6.11 Section Category Options

Item	Command Line Format	Dialog Menu	Specification
Section address	STARt = <sub>[,…] <sub>: [(]<section name> [{ : | , }<section name>[,…]] [)][,...] [/<address>]	Link/Library <Section> [Show entries for :] [Section]	Specifies a section start address
Symbol address file	FSymbol = <section name>[,…]	Link/Library <Section> [Show entries for :] [Symbol file]	Outputs externally defined symbol addresses to a definition file.

STARt Section Address

Link/Library <Section> [Show entries for :] [Section]

Format: STARt = <sub> [,…]

<sub>: [(] <section name> [{ : | , } <section name> [,…]] [)] [,…] [/ <address>]

Description: Specifies the start address of the section. Specify an address as the hexadecimal.

The section name can be specified with wildcards “*”. Sections specified with wildcards are expanded according to the input order.

Two or more sections can be allocated to the same address (i.e., sections are overlaid) by separating them with a colon “:”.

Sections specified at a single address are allocated in the specification order.

Sections to be overlaid can be changed by enclosing them by parentheses “()”.

Objects in a single section are allocated in the specification order of the input file or the input library.

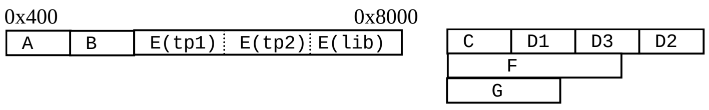
If no address is specified, the section is allocated at 0.

A section which is not specified with the **start** option is allocated after the last allocation address.

Examples: This example shows how sections are allocated when the objects are input in the following order (names enclosed by parentheses are sections in the objects).

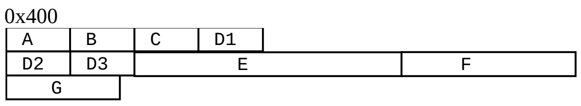
tp1.obj(A,D1,E) -> tp2.obj(B,D3,F) -> tp3.obj(C,D2,E,G)->lib.lib(E)

(1) -start=A,B,E/400,C,D*:F:G/8000



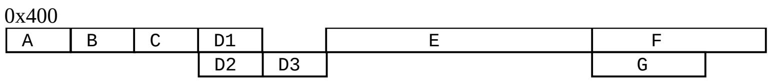
- Sections **C**, **F**, and **G** separated by colons are allocated to the same address.
- Sections specified with wildcards “*” (in this example, the sections whose names start with **D**) are allocated in the input order.
- Objects in the sections having the same name (**E** in this example) are allocated in the input order.
- An input library’s section having the same name (**E** in this example) as those of input objects is allocated after the input objects.

(2) -start=A,B,C,D1:D2,D3,E,F:G/400



- The sections that come immediately after the colons (**A**, **D2**, and **G** in this example) are selected as the start and allocated to the same address.

(3) -start=A,B,C,(D1:D2,D3),E,(F:G)/400



- When the sections to be allocated to the same address are enclosed by parentheses, the sections within parentheses are allocated to the address immediately after the sections that come before the parentheses (**C** and **E** in this example).
- The section that comes after the parentheses (**E** in this example) is allocated after the last of the sections enclosed by the parentheses.

Remarks: When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.

Parentheses cannot be nested.

One or more colons must be written within parentheses. Parentheses cannot be written without a colon.

Colons cannot be written outside of parentheses.

When this option is specified with parentheses, optimization with the linkage editor is disabled.

FSymbol — Symbol Address File

Link/Library <Section> [Show entries for :][Symbol file]

Format: FSymbol = <section name> [,…]

Description: Outputs externally defined symbols in the specified section to a file in the assembler directive format.

The file name is **<output file>.fsy**.

Examples:

```
fSymbol = sct2, sct3
output=test.abs
```

Outputs externally defined symbols in sections **sct2** and **sct3** to **test.fsy**.

[Output example of **test.fsy**]

```
;OPTIMIZING LINKAGE EDITOR GENERATED FILE 1999.11.26
;fsymbol = sct2, sct3

;SECTION NAME = sct2
  .export _f
_f: .equ h'00000000
  .export _g
_g: .equ h'00000016
;SECTION NAME = sct3
  .export _main
_main: .equ h'00000020
  .end
```

Remarks: When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.

This option is available when the CPU type is H8, H8S, H8SX Family , SuperH Family or RX Family.

6.2.6 Verify Options

Table 6.12 Verify Category Options

Item	Command Line Format	Dialog Menu	Specification
Address check	CPu = { <cpu information file name> | <memory type> = <address range>[,…] | STRIDE} <memory type>: { ROm | RAm | XROm | XRAm | YROm | YRAm } <address range>: <start address> -<end address>	Link/Library <Verify> [CPU information check :]	Specifies a specifiable allocation range for section addresses. The specified section will be divided.
Physical space overlap check	PS_check=<sub>[:<sub>...] <sub>: <LS>,<LS>[,...] <LS>: <start address> -<end address>	Link/Library <Verify> [Physical space overlap check :]	Specifies address ranges that may overlap each other in the physical space.
Not divide the specified section	CONTIGUOUS_SECTION = <section name>[,…]	Link/Library <Verify> [Not divide the specified section :]	The specified section will not be divided.

CPu

Address Check

Verify [CPU information check:]

Format: CPu={<cpu information file name>
| <memory type> = <address range> [,…]
| STRIDE}

<memory type>: { ROm | RAm | XROm | XRAm | YROm | YRAm | FIX}

<address range>: <start address> - <end address>

Description: When **cpu=stride** is not specified, a section larger than the specified range of addresses leads to an error.

When **cpu=stride** is specified, a section larger than the specified range of addresses is allocated to the next area of the same memory type or the section is divided.

[Example]
When the **stride** suboption is not specified:
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF
The result is normal when **D1** and **D2** are respectively allocated within the ranges from 100 to 1FF and from 200 to 2FF. If they are not allocated within the ranges, an error will be output.

[Example]
When the **stride** suboption is specified:
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF,ROM=300-3FF
cpu=stride
The result is normal when D1 and D2 are allocated within the ROM area (regardless of whether the section is divided). A linkage error occurs when they are not allocated within the ROM area even though the section is divided.

xrom and **xram** specify the X memory areas and **yrom** and **yram** specify the Y memory areas in the DSP.

Specify an address range in which a section can be allocated in hexadecimal notation. The memory type attribute is used for the inter-module optimization.

FIX for <memory type> is used to specify a memory area where the addresses are fixed (e.g. I/O area).

If the address range of <start>-<end> specified for **FIX** overlaps with that specified for another memory type, the setting for **FIX** is valid.

When <memory type> is **ROM** or **RAM** and the section size is larger than the specified memory range, sub-option **STRIDE** can be used to divide a section and allocate them to another area of the same memory type. Sections are divided in module units.

[Example]
cpu=ROM=0-FFFF,RAM=10000-1FFFF
Checks that section addresses are allocated within the range from 0 to FFFF or from 10000 to 1FFFF. Object movement is not provided between different attributes with the inter-module optimization.

cpu=ROM=100-1FF,ROM=400-4FF,RAM=500-5FF cpu=stride
When section addresses are not allocated within the range from 100 to 1FF, the linkage editor divides the sections in module units and allocates them to the range from 400 to 4FF.

Remarks: When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.

When **cpu=stride** and **memory=low** are specified, its option is unavailable.

Memory types **xrom**, **xram**, **yrom**, and **yram** are available only when the CPU is SHDSP, SH2DSP, SH3DSP or SH4ALDSP.

When **cpu=stride** and **optimize=register** are valid, error L2320 may be output. In such cases, disable **optimize=register**.

When section **B** is divided by **cpu=stride**, the size of section **C$BSEC** increases by 8 bytes x umber of divisions because this amount of information is required for initialization.

PS_check Physical Space Overlap Check

Verify [Physical space overlap check :]

Format:	PS_check=<sub>[:<sub>...]
	<sub>: <LS>,<LS>[,...]
	<LS>: <start address>-<end address>
Description:	Specifies objects that may overlap each other when they are allocated to the memory.
	Use this option to detect SH3 or SH4 objects that will overlap each other when they are allocated to the actual memory even if their virtual addresses do not overlap.
	If an overlap is detected after this option setting, an error will be output and the linkage operation will be terminated.
	Specify address ranges (<LS> in the command line format) that may overlap each other in the memory.
	To check multiple physical memory spaces, specify them by separation with a colon (:).
Examples:	In the SH4, the 4-Gbyte address space is mapped to the 512-Mbyte (29-bit address) external memory area when the MMU is disabled (the upper three bits of address for the 4-Gbyte space are ignored).
	For example, when the **U0** area (00000000 to 0x7fffffff) that can be used in user mode is mapped to the external memory (512 Mbytes), overlapped objects can be detected through the following setting.
	-PS_check=00000000-1fffffff,20000000-3fffffff, 40000000-5fffffff,60000000-7fffffff
	This setting means that addresses 00000000, 20000000, 40000000, and 60000000 are allocated to the same location in the actual memory.
Remarks:	This option is only valid for the SuperH Family CPUs.
	This option is invalid if **object**, **relocate**, or **library** is specified for the **output** format (**form** option).
	This option is invalid when an absolute file is input.
	For the address space specifications of the CPU, refer to the hardware manual of the target CPU.

CONTIGUOUS_SECTION

Not divide the specific section

Link/Library <Verify> [Not divide the specified section :]

Format: CONTIGUOUS_SECTION=<section name>[,...]

Description: Allocates the specified section to another available area of the same memory type without dividing the section when **cpu=stride** is valid.

Examples:

```
start=P,PA,PB/100
cpu=ROM=100-1FF,ROM=300-3FF,ROM=500-5FF
cpu=stride
contiguous_section=PA
```

Section **P** is allocated to address 100.

If section **PA** which is specified as **contiguous_section** is over address 1FF, section **PA** is allocated to address 300 without being divided.

If section **PB** which is not specified as **contiguous_section** is over address 3FF, section **PB** is divided and allocated to address 500.

Remarks: When **cpu=stride** is invalid, this option is unavailable.

6.2.7 Other Options

Table 6.13 Other Category Options

Item	Command Line Format	Dialog Menu	Specification
End code	S9	Link/Library <Other> [Miscellaneous options :] [Always output S9 record at the end]	Always outputs the S9 record.
Stack information file	STACk	Link/Library <Other> [Miscellaneous options :] [Stack information output]	Outputs a stack use information file.
Debugging information compression	Compress NOCOmpress	Link/Library <Other> [Miscellaneous options :] [Compress debug information]	Compresses debugging information Does not compress debugging information
Memory occupancy reduction	MEMory = [High | Low]	Link/Library <Other> [Miscellaneous options :] [Low memory use during linkage]	Specifies the memory occupancy when an input file is loaded
Symbol name modification	REName = <sub>[,…] <sub>: {<file name> (<name>=<name>[,…]) | <module name> (<name><name>[,…]) }	Link/Library <Other> [User defined options :]	Modifies a symbol name or section name.
Symbol name deletion	DELete = <sub>[,…] <sub>: {<module name> | [<file name>] (<name>[,…]) }	Link/Library <Other> [User defined options :]	Deletes a symbol name or module name.
Module replacement	REPlace = <sub>[,…] <sub>: <file> [(<module>[,…])]	Link/Library <Other> [User defined options :]	Replaces modules of the same name in a library file.
Module extraction	EXTract = <module>[,…]	Link/Library <Other> [User defined options :]	Extracts the specified module in a library file.
Debugging information deletion	STRip	Link/Library <Other> [User defined options:]	Deletes debugging information in an absolute file or a library file.
Message level	CHange_message=<sub>[,…] <sub>: {Information | Warning | Error } [=<error number> [-<error number>] [,…]]	Link/Library <Other> [User defined options:]	Modifies message levels.
Local symbol name hide	Hide	Link/Library <Other> [User defined options:]	Deletes local symbol name information
Showing total sizes of sections	Total_size	Link/Library <Other> [Miscellaneous options :] [Displays total section size]	This newly added option sends total sizes of sections after linkage to standard output.
Information file for the emulator	RTs_file	Link/Library <Other> [Miscellaneous options :] [Rts information output]	Outputs an information file for the emulator (for SuperH Family).

S9 — End Code

Link/Library <Other>[Miscellaneous options :][Always output S9 record at the end]

Format:	S9
Description:	Outputs the **S9** record at the end even if the entry address exceeds 0x10000.
Remarks:	When **form=stype** is not specified, this option is unavailable.

STACk — Stack Information File

Link/Library <Other>[Miscellaneous options :][Stack information output]

Format:	STACk
Description:	Outputs a stack consumption information file. The file name is <output file name>.sni.
Remarks:	When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.

COmpress, NOCOmpress — Debugging Information Compression

Link/Library <Other>[Miscellaneous options :][Compress debug information]

Format:	COmpress NOCOmpress
Description:	Specifies whether debugging information is compressed. When **compress** is specified, the debugging information is compressed. When **nocompress** is specified, the debugging information is not compressed. By compressing the debugging information, the debugger loading speed is improved. If the **nocompress** option is specified, the link time is reduced. If this option is omitted, the default is **nocompress**.
Remarks:	When form={object | relocate | library | hexadecimal | stype | binary} or strip is specified, this option is unavailable.

MEMory — Memory Occupancy Reduction

Link/Library <Other>[Miscellaneous options :][Low memory use during linkage]

Format: MEMory = [High | Low]

Description: Specifies the memory size occupied for linkage.

When **memory = high** is specified, the processing is the same as usual.

When **memory = low** is specified, the linkage editor loads the information necessary for linkage in smaller units to reduce the memory occupancy. This increases file accesses and processing becomes slower when the occupied memory size is less than the available memory capacity.

memory = low is effective when processing is slow because a large project is linked and the memory size occupied by the linkage editor exceeds the available memory in the machine used.

Remarks: When one of the following options is specified, this option is unavailable:
optimize, compress, delete, rename, map, stack, replace, and
combination of list and show[={reference | xreference}]

Some combinations of this option and the input or output file format are unavailable. For details, refer to Table 6.4 in section 5.2.2, Output Options.

REName — Symbol Name Modification

Link/Library <Other>[User defined options :]

Format: REName = <suboption> [,…]

<suboption>: {[<file>] (<name> = <name> [,…])
| [<module>] (<name> = <name> [,…]) }

Description: Modifies a symbol name or a section name.

Symbol names or section names in a specific file or library in a module can be modified.

For a C/C++ variable name, add an underscore (_) at the head of the definition name in the program.

When a function name is modified, the operation is not guaranteed.

If the specified name matches both section and symbol names, the symbol name is modified.

If there are several files or modules of the same name, the priority depends on the input order.

Examples: rename=(_sym1=data) ; Modifies sym1 to data.

rename=lib1(P=P1) ; Modifies the section **P** to **P1**
; in the library module **lib1**.

Remarks: When **extract** or **strip** is specified, this option is unavailable.

When **form=absolute** is specified, the section name of the input library cannot be modified.

DELete — Symbol Name Deletion

Link/Library <Other>[User defined options :]

Format: DELete = <suboption> [,…]

<suboption>: {[<file>] (<name>[,...]) | <module>}

Description: Deletes an external symbol name or library module.

Symbol names or modules in the specified file can be deleted.

For a C/C++ variable name or C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function name, enclose the definition name in the program with double-quotes including the parameter strings. If the parameter is **void**, specify as "<function name>()". If there are several files or modules of the same name, the file that is input first is applied.

When a symbol is deleted using this option, the object is not deleted but the attribute is changed to the internal symbol.

Examples:

```
delete=(_sym1)          ; Deletes the symbol _sym1 in all files.

delete=file1.obj(_sym2)         ; Deletes the symbol _sym2
; in the input file file1.obj.
```

Remarks: When **extract** or **strip** is specified, this option is unavailable.

REPlace — Module Replacement

Link/Library <Other>[User defined options :]

Format: REPlace = <suboption> [,…]

<suboption>: <file name> [(<module name> [,…]) }

Description: Replaces library modules.

Replaces the specified file or library module with the module of the same name in the library specified with the **library** option.

Examples:

```
replace=file1.obj       ; Replaces the module file1
                        ; with the module file1.obj.

replace=lib1.lib(mdl1)  ; Replaces the module mdl1 with
                        ; the module mdl1 in the input library
                        ; file lib1.lib.
```

Remarks: When **form={object | relocate | absolute | hexadecimal | stype | binary}**, **extract**, or **strip** is specified, this option is unavailable.

EXTract — Module Extraction

Link/Library <Other>[User defined options :]

Format: EXTract = <module name> [,…]

Description: Extracts library modules.

Extracts the specified library module from the library file specified using the **library** option.

Examples: extract=file1 ; Extracts the module **file1**.

Remarks: When **form={absolute | hexadecimal | stype | binary}** or **strip** is specified, this option is unavailable.

STRip — Debugging Information Deletion

Link/Library <Other>[User defined options :]

Format: STRip

Description: Deletes debugging information in an absolute file or library file.

When the **strip** option is specified, one input file should correspond to one output file.

Examples: input=file1.abs file2.abs file3.abs
strip

Deletes debugging information of **file1.abs**, **file2.abs**, and **file3.abs**, and outputs this information to **file1.abs**, **file2.abs**, and **file3.abs**, respectively. Files before debugging information is deleted are backed up in **file1.abk**, **file2.abk**, and **file3.abk**.

Remarks: When **form={object | relocate | hexadecimal | stype | binary}** is specified, this option is unavailable.

CHange_message — Message Level

Link/Library <Other>[User defined options :]

Format: CHange_message = <suboption> [,…]

<suboption>: <error level> [= <error number> [-<error number>] [,…]]

<error level>: {Information | Warning | Error}

Description: Modifies the level of information, warning, and error messages.

Specifies the execution continuation or abort at the message output.

Examples: change_messag=warning=2310
Modifies L2310 to the warning level and specifies execution continuation at L2310 output.

change_message=error
Modifies all information and warning messages to error level messages.
When a message is output, the execution is aborted.

Hide Local Symbol Name Hide

Link/Library <Other>[User defined options :]

Format: Hide

Description: Deletes local symbol name information from the output file. Since all the name information regarding local symbols is deleted, local symbol names cannot be checked even if the file is opened with a binary editor. This option does not affect the operation of the generated file.

Use this option to keep the local symbol names secret.

The following types of symbol names are hidden:
C source: Variable or function names specified with the **static** qualifiers
C source: Label names for the **goto** statements
Assembly source: Symbol names of which external definition (reference) symbols are not declared

* The entry function name is not hidden.

Examples: The following is a C source example in which this option is valid:

```
int g1;
int g2=1;
const int g3=3;
static int s1;            //<- The static variable name will be hidden.
static int s2=1;          //<- The static variable name will be hidden.
static const int s3=2;    //<- The static variable name will be hidden.

static int sub1()         //<- The static function name will be hidden.
{
    static int s1;        //<- The static variable name will be hidden.
    int l1;

    s1 = l1; l1 = s1;
    return(l1);
}

int main()
{
    sub1();
    if (g1==1)
        goto L1;
    g2=2;
L1:                       //<- The label name of the goto statement
                          //    will be hidden.
    return(0);
}
```

Remarks: This option is available only when the output file format is specified as **absolute**, **relocate**, or **library**.

When the input file was compiled or assembled with the **goptimize** option specified, this option is unavailable if the output file format is specified as **relocate** or **library**.

To use this option with the external variable access optimization, do not use this option for the first linkage, and use it only for the second linkage.

The symbol names in the debugging information are not deleted by this option.

Total_size

Showing total sizes of sections

Link/Library <Other> [Miscellaneous options :] [Displays total section size]

Format: Total_size

Description: Sends total sizes of sections after linkage to standard output. The sections are categorized as follows, with the overall size of each being output.

- Executable program sections
- Non-program sections allocated to the ROM area
- Sections allocated to the RAM area

This option makes it easy to see the total sizes of sections allocated to the ROM and RAM areas.

Remarks: The **show=total_size** option must be used if total sizes of sections are to be output in the linkage listing.

When the ROM-support function (**rom** option) has been specified for a section, the section will be used by both the source (ROM) and destination (RAM) of the transfer. The sizes of sections of this type will be added to the total sizes of sections in both ROM and RAM.

RTs_file

Information File for the Emulator

Link/Library <Other> [Miscellaneous options :] [Rts information output]

Format: RTs_file

Description: This option creates a return address information file (**.rts** file) for the emulator. For usage of this option, refer to the user’s manual for the emulator in use. This option is not available in some types of emulators.

The name of the return address information file is **<load module name>.rts**. If the file to be output is **test.abs** as specified with the **output** option, for example, its file will be created as **test.rts**. The return address information file is created under the same directory where the load module has been created.

Remarks: This option is invalid when **form={object | relocate | library}** has been specified.

This option is invalid when an absolute file is selected as an input file.

For usage of this option, refer to the user’s manual for the emulator in use. This option is not available in some types of emulators.

This option can be used when the CPU type is SuperH Family.

6.2.8 Subcommand File Options

Table 6.14 Subcommand Tab Option

Item	Command Line Format	Dialog Menu	Specification
Subcommand file	SUbcommand = <file name>	Link/Library <Subcommand file> [Use external subcommand file]	Specifies options with a subcommand file

SUbcommand Subcommand File

Link/Library <Subcommand file> [Use external subcommand file]

Format: SUbcommand = <file name>

Description: Specifies options with a subcommand file.

The format of the subcommand file is as follows:
<option> { = | Δ } [<suboption> [,…]] [Δ&] [;<comment>]

The option and suboption are separated by an “=” sign or a space.

For the **input** option, suboptions are separated by a space.

One option is specified per line in the subcommand file.

If a subcommand description exceeds one line, the description can be allowed to overflow to the next line by using an ampersand (&).

The **subcommand** option cannot be specified in the subcommand file.

Examples: Command line specification:

```
optlnk file1.obj -sub=test.sub file4.obj
```

Subcommand specification:

```
input     file2.obj file3.obj          ; This is a comment.
library lib1.lib, &                    ; Specifies line continued.
lib2.lib
```

Option contents specified with a subcommand file are expanded to the location at which the subcommand is specified on the command line and are executed.

The order of file input is **file1.obj**, **file2.obj**, **file3.obj**, and **file4.obj**.

6.2.9 CPU Option

Table 6.15 CPU Tab Option

Item	Command Line Format	Dialog Menu	Specification
SBR address specification	SBr = { <SBR address> | User}	CPU [Specify SBR address :]	Specifies the start address of the 8-bit absolute area (for H8SX Family).

SBr — SBR Address Specification

CPU [Specify SBR address :]

Format: SBr = { <address> | User }

Description: Specifies the **SBR** address.

When the **SBR** address is specified in this option, optimization using the **abs8** area is available. When **user** is specified in this option, optimization for the **abs8** area is disabled.

Remarks: This option is available only when the CPU is H8SX Family.

If more than one **SBR** address is specified within the source or by tool options, the optimizing linkage editor assumes that **user** is specified regardless of this option setting.

6.2.10 Options Other Than Above

Table 6.16 Options Other Than Above

Item	Command Line Format	Dialog Menu	Specification
Copyright	<u>LOgo</u> NOLOgo	—	Output Not output
Continuation	END	—	Executes option strings already input, inputs continuing option strings and continues processing.
Termination	EXIt	—	Specifies the termination of option input.

LOgo, NOLOgo — Copyright

None (nologo is always available.)

Format: <u>LOgo</u>

NOLOgo

Description: Specifies whether the copyright is output.

When the **logo** option is specified, the copyright is displayed.

When the **nologo** option is specified, the copyright display is disabled.

When this option is omitted, the default is **logo**.

END — Execution Continued

None

Format: END

Description: Executes option strings specified before **END**. After the linkage processing is terminated, option strings that are specified after **END** are input and the linkage processing is continued.

This option cannot be specified on the command line.

Examples:

```
input=a.obj,b.obj          ; Processing (1)
start=P,C,D/100,B/8000     ; Processing (2)
output=a.abs               ; Processing (3)
end
input=a.abs                ; Processing (4)
form=stype                 ; Processing (5)
output=a.mot               ; Processing (6)
```

Executes the processing from (1) to (3) and outputs **a.abs**. Then executes the processing from (4) to (6) and outputs **a.mot**.

EXIt — Termination Processing

None

Format: EXIt

Description: Specifies the end of the option specifications.

This option cannot be specified on the command line.

Examples: Command line specification:

```
optlnk -sub=test.sub -nodebug
```

test.sub:

```
    input=a.obj,b.obj          ; Processing (1)
    start=P,C,D/100,B/8000     ; Processing (2)
    output=a.abs               ; Processing (3)
    exit
```

Executes the processing from (1) to (3) and outputs **a.abs**.

The **nodebug** option specified on the command line after **exit** is executed is ignored.

Section 7 Environment Variables

7.1 Environment Variables

Environment variables are listed in table 7.1.

Table 7.1 Environment Variables

No.	Environment Variable	Description	Default When Specification is Omitted
1	path	Specifies a storage folder for the execution file	Specification cannot be omitted.
2	BIN30	Specifies the folder in which the executable files of the compiler, assembler, optimizing linkage editor, etc. are stored.	Specification cannot be omitted.
3	INC30	Specifies the folder in which the standard include files of the compiler and assembler are stored.	Specification cannot be omitted.
4	LIB30	Specifies the folder in which the standard libraries and internal tools of the compiler or assembler are stored.	Specification cannot be omitted.
5	TMP30	Specifies a directory in which a temporary file is generated.	Specification cannot be omitted.
6	HLNK_LIBRARY1 HLNK_LIBRARY2 HLNK_LIBRARY3	Specifies a default library name for the optimizing linkage editor. Libraries which are specified by a library option are linked first. Then, if there is an unresolved symbol, the default libraries are searched in the order of 1, 2, 3.	No value is set when specification is omitted.
7	HLNK_TMP	Specifies a folder in which the optimizing linkage editor generates temporary files. If HLNK_TMP is not specified, the temporary files are created in the current folder.	No value is set when specification is omitted.
8	HLNK_DIR	Specifies an input file storage folder for the optimizing linkage editor.	No value is set when specification is omitted.

- When more than one directory is specified by INC30, HLLNK_LIBRARY1, HLLNK_LIBRARY2, HLNK_LIBRARY3, and HLNK_DIR, the directories should be divided using semicolons (;).
- For folder specification, specify the one that has access rights.
- These environment variables can be set easily by executing the batch file setnc30.bat which is generated at installation. setnc30.bat is stored in "<High-performance Embedded Workshop storage directory> \Tools\Renesas\nc30wa\<nc30wa storage directory>\..".

7.2 Predefined Macros

The following symbol constant setting options are set according to the option specification and version.

Table 7.2 Symbol Constant Setting Options

No.	Option	Symbol Constant Setting Options
1	-R8C, -R8CE, -R8Cxx	-D__R8C__=1

Section 8 File Specifications

8.1 Naming Files

A standard file extension is automatically added to the name of a compiled file when omitted. The standard file extensions used by the integrated development environment are shown in table 8.1.

Table 8.1 Standard File Extensions Used by the Integrated Development Environment

No.	File Extension	Description
1	a30	Assembler source file
2	inc	Assembler include file
3	lst	Assembler list file
4	atg	Assembler error tag file
5	obj	Relocatable object file
6	abs	Absolute load module file
7	map	Linkage map list file
8	id	ID file
9	lib	Library file
10	lbp	Library list file
11	mot	S-type format file
12	hex	HEX format file
13	bin	Binary file
14	sni	Stack information file
15	pro	Profile information file
16	dbg	Debugging information file
17	rti	Object file including definition that is specified by a file with extension td
18	cal	Information file to be called
19	bls	Information file for external symbol allocation
20	utl	Utl30 information file
21	rel	Relocatable file

File names beginning with rti_ are reserved for the system; do not use those file names.

8.2 Assembler source file

8.2.1 Source file format

The source files are created in text format. Use a text editor, etc. to write source files following Chapter 3, "Assembler Language Description Rules".

8.2.2 Source file name

Specify any source file name. In this assembler, the source file extension is, by default, ".a30". If a file name is defined with other than this extension, specify the file with a full name when activating the assembler.

8.3 Assembler include file

8.3.1 Include file format

Use a text editor, etc. to write include files following "Rules for Writing a Program".

8.3.2 Include file name

Specify any include file name. In this assembler, the include file extension is, by default, ".inc". If a file name is defined with other than this extension, specify a full name in the source line where the include file is specified.

8.4 Assembler list file

8.4.1 Structure of Assembler List

The assembler list file shows information on assembly results.
The composition and the content of the assembler list file are shown in Table 8.2.

Table 8.2 Structure and Contents of Assembler List

No.	Information shown in list file	Contents	When option "-H"specified
1	List header information	Assembler list file created date/time and pages and header information on objects	Not output
2	Object information	Object code and source code	Output
3	Statistics information	Total number of errors, number of source program lines, and section size	Output

Note: The option "-H" is valid when the option "-L" is specified.

8.4.2 List header information

List header information is output to the assembler list file by default.
Note, however, that this information is not output when the option "-H" is specified.
For example output, see Figure 8.1, "Example Output of Assembler List Files".

8.4.3 Object information

An example output of object information is shown in Figure 8.1, "Example Output of Assembler List Files".

```
* M16C Series and R8C Family Assmbler *    SOURCE LIST        Tue Jun 1 12:34:56 2010   PAGE 001

  SEQ.   LOC.    OBJ.               0XMSDA ....*....SOURCE STATEMENT....7....*....8....*....9....
  (1)    (2)      (3)                 (4)      (5)                                  Header information
1                                              ;
2                                              ;   AS30 sample source file
3                                              ;
4                                              ;-----------------------------------
5
6                                              ; Macro define
7                                       D      mac1    .MACRO         p1,p2
8                                       D          MOV.W      p1,p2
9                                       D          MOV.W      p1,R2
10                                      D          MOV.W      p2,R3
11                                             .ENDM
12
13                                                 .SECTION          ram1,data
14   00000(000001H)                            work1:        .BLKB      1
15   00001(000001H)                            work2:        .BLKB      1
16
17   00000001h                                 sym1          .EQU       1
18   00000002h                                 sym2          .EQU       2
19
20                                                 .SECTION          prog1,code
21   00000                                     samp_start:
22
23                                             .IF           MODE == 1
24                                    X             MOV.B    #sym1,R0L
25                                             .ELIF         MODE == 2
26                                    X            MOV.B    #sym2,R0L
27                                             .ELSE
28   00000   B4                           Z        MOV.B    #0,R0L
29                                             .ENDIF
30
31   00001                                      ..tl0001:
32   00001   E301                         S        CMP.B    #sym1,R0H
33   00003   6A04                                  JEQ       ..tl0002
34   00005   D802                         Q        MOV.B    #0,R1L
35   00007   61                           S        JMP       lab1
36   00008                                     ..tl0002:
37   00008   D803                         Q        MOV.B    #0,R1H
38   0000A                                     lab1:
39   0000A   D91F0000r                    Q        MOV.W    #sym1,work1
40   0000E   D92F0000r                    Q        MOV.W    #sym2,work2
41                                                mac1       R0,12h
42   00012   730F1200                 M           MOV.W     R0,12h
43   00016   7302                     M           MOV.W     R0,R2
44   00018   73F31200                 M           MOV.W     12h,R3
45                                    M            .ENDM
46                                                 .END
```

Figure 8.1 Example Output of Assembler List Files.

(1) List line information : SEQ.
Outputs the line numbers of the assembler list.

(2) Location information : LOC.
Outputs the location addresses of a range of object code that can be determined when assembling.

(3) Object code information : OBJ.
Outputs the object code corresponding to mnemonics.

(4) Line information : 0XMSDA
Outputs information on the results of source line processing performed by as30.
Specifically, this information contains the following:

Table 8.3 Line information of Assembler List

0	X	M	S	D	A	Contents
0-9						Indicates the include file's nest rebel.
	X					Indicates that this line was not assembled in condition assemble.
		M				Indicates that this is a macro expansion line.
		D				Indicates that this is a macro definition line.
			S			Indicates that this is a structured description expansion line.
				S		Indicates that jump distance specifier S was selected.
				B		Indicates that jump distance specifier B was selected.
				W		Indicates that jump distance specifier W was selected.
				A		Indicates that jump distance specifier A was selected.
				Z		Indicates that zero form (:Z) was selected for the instruction format.
				S		Indicates that short form (:S) was selected for the instruction format.
				Q		Indicates that quick form (:Q) was selected for the instruction format.
					*	Indicates that 8-bit displacement SB relative addressing mode was selected.

(5) Source line information : ....*....SOURCE STATEMENT....
Outputs the assembly source line.

8.4.4 Statistics Information

Figure 8.2 shows an example of statistics information output.

```
Information List     (1)

TOTAL ERROR(S)      00000
TOTAL WARNING(S)    00000
TOTAL LINE(S)       00046  LINES

Section List         (2)

Attr    Size              Name
DATA    0000002(00002H)   ram1
CODE    0000028(0001CH)   prog1
```

Figure 8.2 Example of Statistics Information Output

(1) Numbers of error messages and warning messages, and total number of source lines

(2) Section information (section attribute, size, and section name)

8.5 Assembler error tag file

Only when you specified command options (-T and -X), as30 outputs to a file the errors that were encountered when assembling the assembly source file.

(1) File name of assembler error tag file

The file name of the assembler error tag file is created by changing the extension of the assembly source file (.a30 by default) to ".atg". (sample.a30 --> sample.atg)

(2) Directory for assembler error tag file generated

If you specified the directory with command option (-O), the assembler error tag file is generated in that directory. If no directory is specified, the assembler error tag file is generated in the directory where the assembly source file resides.

8.6 Linkage List

This section covers the contents and format of the linkage list output by the optimizing linkage editor.

8.6.1 Structure of Linkage List

Table 8.4 shows the structure and contents of the linkage list.

Table 8.4 Structure and Contents of Linkage List

No.	Output Information	Contents	When show Option* is Specified	When show Option is not Specified
1	Option information	Option strings specified by a command line or subcommand	None	Output
2	Error information	Error messages	None	Output
3	Linkage map information	Section name, start/end addresses, size, and type	None	Output
4	Symbol information	Static definition symbol name, address, size, and type in the order of address	show =symbol	Not output
		When show=reference is specified: Symbol reference count and optimization information in addition to the above information	show =reference	Not output
5	Symbol deletion optimization information	Symbols deleted by optimization	show =symbol	Not output
6	Cross-reference information	Symbol reference information	show =xreference	Not output
7	Total section size	Total sizes of RAM, ROM, and program sections	show=total_size	Not output
8	Vector information	Vector numbers and address information	show=vector	Not output
9	CRC information	CRC calculation result and output addresses	None	Always output when the CRC option is specified

Note: *The show option is valid when the list option is specified.

8.6.2 Option Information

The option strings specified by a command line or a subcommand file are output. Figure 8.3 shows an example of option information output when optlnk -sub=test.sub -list -show is specified.

```
(test.sub contents)
INPUT test.obj
```

```
*** Options ***

-sub=test.sub          \
INPUT test.obj (2)      } (1)
-list                   |
-show                  /
```

Figure 8.3 Example of Option Information Output (Linkage List)

(1) Outputs option strings specified by a command line or a subcommand in the specified order.

(2) Subcommand in the test.sub subcommand file

8.6.3 Error Information

Error messages are output. Figure 8.4 shows an example of error information output.

```
*** Error Information ***
                                                                     \
** L2310 (E) Undefined external symbol “strcmp” referred to in “test.obj” } (1)
```

Figure 8.4 Example of Error Information Output (Linkage List)

(1) Outputs an error message.

8.6.4 Linkage Map Information

The start and end addresses, size, and type of each section are output in the order of address. Figure 8.5 shows an example of linkage map information output.

```
*** Mapping List ***

SECTION                                    START      END         SIZE   ALIGN
  (1)                                       (2)       (3)          (4)    (5)

P
                                         00001000  00001000          1    1
C
                                         00001004  00001007          4    4
D_2
                                         00001008  000014dd        4d6    2
B_2
                                         000014de  000050b3       3bd6    2
```

Figure 8.5 Example of Linkage Map Information Output (Linkage List)

(1) Section name

(2) Start address

(3) End address

(4) Section size

(5) Section boundary alignment value

8.6.5 Symbol Information

When show=symbol is specified, the addresses, sizes, and types of externally defined symbols or static internally defined symbols are output in the order of address. When show=reference is specified, the symbol reference counts and optimization information are also output. Figure 8.6 shows an example of symbol information output.

```
*** Symbol List ***

SECTION=(1)
FILE=(2)        START          END            SIZE
                 (3)           (4)             (5)
  SYMBOL        ADDR           SIZE           INFO             COUNTS  OPT
  (6)            (7)           (8)             (9)              (10)   (11)

SECTION=P
FILE=test.obj
                00000000       00000428            428
  _main
                00000000              2       func ,g               0
  _malloc
                00000000             32       func ,l               0
FILE=mvn3
                00000428       00000490             68
  $MVN#3
                00000428              0       none ,g               0
```

Figure 8.6 Example of Symbol Information Output (Linkage List)

(1) Section name

(2) File name

(3) Start address of a section included in the file indicated by (2) above

(4) End address of a section included in the file indicated by (2) above

(5) Section size of a section included in the file indicated by (2) above

(6) Symbol name

(7) Symbol address

(8) Symbol size

(9) Symbol type as shown below

Data type:	func	Function name
	data	Variable name
	entry	Entry function name
	none	Undefined (label, assembler symbol)
Declaration type:	g	External definition
	l	Internal definition

(10) Symbol reference count only when show=reference is specified. * is output when show=reference is not specified.

(11) Optimization information as shown below.

ch	Symbol modified by optimization
cr	Symbol created by optimization
mv	Symbol moved by optimization

8.6.6 Symbol Deletion Optimization Information

The size and type of symbols deleted by symbol deletion optimization (optimize=symbol_delete) are output. Figure 8.7 shows an example of symbol deletion optimization information output.

```
*** Delete Symbols ***

SYMBOL                           SIZE    INFO
  (1)                            (2)     (3)
  _Version
                                    4    data ,g
```

Figure 8.7 Example of Symbol Deletion Optimization Information Output (Linkage List)

(1) Deleted symbol name

(2) Deleted symbol size

(3) Deleted symbol type as shown below

Data type:	func	Function name
	data	Variable name
Declaration type:	g	External definition
	l	Internal definition

8.6.7 Cross-Reference Information

When show=xreference is specified, symbol reference information (cross reference information) is output. Figure 8.8 shows an example of cross-reference information output.

```
 *** Cross Reference List ***

No    Unit Name  Global.Symbol   Location    External Information
(1)     (2)           (3)           (4)               (5)
0001  a
      SECTION=P  _func
                                 00000100
                 _func1
                                 00000116
                 _main
                                 0000012c
                 _g
                                 00000136
      SECTION=B
                 _a
                                 00000190    0001(00000140:P)
                                             0002(00000178:P)
                                             0003(0000018c:P)
0002  b
      SECTION=P
                 _func01
                                 00000154    0001(00000148:P)
                 _func02
                                 00000166    0001(00000150:P)
0003  c
      SECTION=P
                 _func03
                                 00000184
```

Figure 8.8 Example of Cross-Reference Information Output (Linkage List)

(1) Unit number, which is an identification number in object units

(2) Object name, which specifies the input order at linkage

(3) Symbol name output in ascending order of allocation addresses for every section

(4) Symbol allocation address, which is a relative value from the beginning of the section when form=rel is specified

(5) Address of an external symbol that has been referenced
Output format: <Unit number> (<address or offset in section>:<section name>)

8.6.8 Total Section Size

When show=total_size is specified, total section size is output. Figure 8.9 shows an example of total section size output.

```
*** Total Section Size  ***

RAMDATA SECTION:        00000660 Byte(s)
(1)
ROMDATA SECTION:        00000174 Byte(s)
(2)
PROGRAM SECTION:        000016d6 Byte(s)
(3)
```

Figure8.9 Example of Total Section Size Output (Linkage List)

(1) Total size of RAM data sections

(2) Total size of ROM data sections

(3) Total size of program sections

8.6.9 Variable Vector Table Information

When show=vector is specified, variable vector table is output. Figure 8.10 shows an example of variable vector table output.

```
*** Variable Vector Table List ***

No.    SYMBOL/ADDRESS
(1)       (2)
  0       __brk
  1       __dummy_int
  2       __dummy_int
  3       __dummy_int
  4       __int3
  5       __timer_b5
  :
  <Omitted>
```

Figure 8.10 Example Output of a Variable Vector Table (Linkage List)

(1) Variable vector numbers

(2) Shows symbols. If no symbols are defined, this list is indicated with addresses.

8.6.10 Special Page Vector Table Information

When show=vector is specified, special page vector table is output. Figure 8.11 shows an example of special page vector table output.

```
*** Special Vector Table List ***

NO.    SYMBOL/ADDRESS
(1)    (2)
20     __sfunc20
19     __sfunc19
18     __sfunc18
```

Figure 8.11 Example Output of a Special Page Vector Table (Linkage List)

(1) Special page vector numbers.

(2) Shows symbols. If no symbols are defined, this list is indicated with addresses.

8.6.11 ID code, Protect code and OFSREG code Information

The contents of the ID code, Protect code and OFSREG code are output. Figure 8.12 shows an example of each information output.

```
*** ID code information ***  (1)

CHARACTOR STRING="sample"
NUMERICAL VALUE=
  0000ffdf:   73
  0000ffe3:   61
  0000ffeb:   6d
  0000ffef:   70
  0000fff3:   6c
  0000fff7:   65
  0000fffb:   00

*** Protect code or OFSREG code information ***  (2)
  0000ffff:   ff
```

Figure 8.12 Example of ID code, Protect code and OFSREG code Output (Linkage List)

(1) ID code information

(2) Protect code or OFSREG code

8.7 ID file

When you specify the assembler directive command(.ID), optlnk outputs ID code to a file.
The contents of the ID code are output. Figure 8.13.1 and Figure 8.13.2 shows an example of ID file output.

```
-IDsample -protectx FF
FFFDF : 73
FFFE3 : 61
FFFEB : 6D
FFFEF : 70
FFFF3 : 6C
FFFF7 : 65
FFFFB : 00
FFFFF : FF
```

Figure 8.13.1 Example of ID file Output

```
-IDsample -ofsregx FF
0FFDF : 73
0FFE3 : 61
0FFEB : 6D
0FFEF : 70
0FFF3 : 6C
0FFF7 : 65
0FFFB : 00
0FFFF : FF
```

Figure 8.13.2 Example of ID file Output(When –R8C,-R8CE ot –R8Cxx Option is Specified)

8.8 Library List

This section covers the contents and format of the library list output by the optimizing linkage editor.

8.8.1 Structure of Library List

Table 8.5 shows the structure and contents of the library list.

Table 8.5 Structure and Contents of Library List

No.	Output Information	Contents	Suboption *	When show Option is not Specified
1	Option information	Option strings specified by a command line or subcommand	—	Output
2	Error information	Error messages	—	Output
3	Library information	Library information	—	Output
4	Information of modules, sections, and symbols within library	Module within the library	—	Output
		When show=symbol is specified: List of symbol names in a module within the library	show=symbol	Not output
		When show=section is specified: Lists of section names and symbol names in a module within the library	show=section	Not output

Note: *All options are valid when the list option is specified.

8.8.2 Option Information

The option strings specified by a command line or a subcommand file are output. Figure 8.14 shows an example of option information output when optlnk –sub = test.sub -list -show is specified.

```
(test.sub contents)
form    library
in      adhry.obj
output  test.lib
```

```
*** Options ***

-sub=test.sub           \
form    library  \       |
in      adhry.obj }(2)   } (1)
output  test.lib /       |
-list                    |
-show                   /
```

Figure 8.14 Example of Option Information Output (Library List)

(1) Outputs option strings specified by a command line or a subcommand in the specified order.

(2) Subcommand in the test.sub subcommand file

8.8.3 Error Information

Messages for errors or warnings are output. Figure 8.15 shows an example of error information output.

```
*** Error Information ***

** L1200 (W) Backed up file “main.lib” into “main.lbk”    (1)
```

Figure 8.15 Example of Error Information Output (Library List)

(1) Outputs a warning message.

8.8.4 Library Information

The library type is output. Figure 8.16 shows an example of library information output.

```
*** Library Information  ***

LIBRARY NAME=test.lib   (1)
CPU=SuperH              (2)
ENDIAN=Big              (3)
ATTRIBUTE=system        (4)
NUMBER OF MODULE=1      (5)
```

Figure 8.16 Example of Library Information Output (Library List)

(1) Library name

(2) CPU name

(3) Endian type

(4) Library file attribute: either system library or user library

(5) Number of modules within the library

8.8.5 Module, Section, and Symbol Information within Library

A list of modules within the library is output.

When show=symbol is specified, the symbol names in a module within the library are listed. When show=section is specified, the section names and symbol names in a module within the library are listed.

Figure 8.17 shows an output example of module, section, and symbol information within a library.

```
*** Library List ***

MODULE          LAST UPDATE
 (1)                (2)
  SECTION
    (3)
    SYMBOL
      (4)
adhry
                29-Feb-2000 12:34:56
  P
   _main
   _Proc0
   _Proc1
  C
  D
   _Version
  B
   _IntGlob
   _CharGlob
```

Figure 8.17 Example of Module, Section, and Symbol Information Output (Library List)

(1) Module name

(2) Module definition date
If the module is updated, the latest module update date is displayed.

(3) Section name within a module

(4) Symbol within a section

Section 9 Assembler directive commands

9.1 Address Control Directive Commands

These directive commands control address specifications in the assembler.

The assembler handles relocatable address values except for the addresses in absolute-addressing sections.

Table 9.1 Address control directive commands

Directive	**Function**
.ORG	Declares the start address. The section including this directive becomes an absolute-addressing section.
.BLKB	Allocates a RAM area in 1-byte units.
.BLKW	Allocates a RAM area in 2-byte units.
.BLKA	Allocates a RAM area in 3-byte units.
.BLKL	Allocates a RAM area in 4-byte units.
.BLKF	Allocates a RAM area in 4-byte units.
.BLKD	Allocates a RAM area in 8-byte units.
.BYTE	Stores 1-byte data in a ROM area.
.WORD(S)	Stores 2-byte data in a ROM area.
.ADDR	Stores 3-byte data in a ROM area.
.LWORD	Stores 4-byte data in a ROM area.
.FLOAT	Stores floating-point data represented by four bytes in a ROM area.
.DOUBLE	Stores floating-point data represented by eight bytes in a ROM area.
.ALIGN	Corrects a location counter to a multiple of the boundary alignment value.

Address Declaration

.ORG

Format: [Δ].ORGΔ<operand>

Description This directive command, when written immediately after the section definition directive command ".SECTION", makes the relevant section assume the absolute attribute.

This directive command can be written multiple times in an absolute-attribute section.

In relative-attribute sections, this directive command cannot be written.

Examples:

```
.SECTION     value,ROMDATA
.ORG         0FF00H
.BYTE        "abcdefghijklmnopqrstuvwxyz"
.ORG         0FF80H
.BYTE        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

The example shown below will cause an error, because this directive command is used in a relative-attribute section.

```
.SECTION     value,ROMDATA
.ORG         0FF00H
.BYTE        "abcdefghijklmnopqrstuvwxyz"
.ORG         0FF80H
.BYTE        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

Remarks: The values writable in the operand are numeric values in the range 0 to 0FFFFFFH (or in the range 0 to 0FFFFFH, if the -R8C option is specified).

An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Notes: Absolute-attribute sections cannot have their addresses relocated at link time.

Unless ".ORG" is written in the line next to the one where the section definition directive command ".SECTION" is written, the section is assumed to be a relative-attribute section.

If a plurality of ".ORG" are written in a section with the same name as a CODE or ROMDATA type of section, codeless blank spaces in it are filled with NOP instructions (04H).

1-Byte Area Allocation

.BLKB

Format:	[Δ][<label name:>Δ].BLKBΔ<operand>
Description	Reserves as many bytes of RAM area as specified by the operand in 1-byte units. An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Examples:

```
symbol    .EQU       1
          .SECTION   area,DATA
work1:    .BLKB      1
work2:    .BLKB      symbol
          .BLKB      symbol+1
```

Notes: Be sure that this directive command is written in a section of DATA type.

Be sure to write a colon (:) for the label name.

2-Byte Area Allocation

.BLKW

Format: [Δ][<label name:>Δ].BLKWΔ<operand>

Description Reserves as many bytes of RAM area as specified by the operand in 2-byte units.

An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Examples:

```
symbol    .EQU       1
          .SECTION   area,DATA
work1:    .BLKW      1
work2:    .BLKW      symbol
          .BLKW      symbol+1
```

Notes: Be sure that this directive command is written in a section of DATA type.

Be sure to write a colon (:) for the label name.

3-Byte Area Allocation

.BLKA

Format: [Δ][<label name:>Δ].BLKAΔ<operand>

Description Reserves as many bytes of RAM area as specified by the operand in 3-byte units.

An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Examples:

```
symbol   .EQU      1
         .SECTION  area,DATA
work1:   .BLKA     1
work2:   .BLKA     symbol
         .BLKA     symbol+1
```

Notes: Be sure that this directive command is written in a section of DATA type.

Be sure to write a colon (:) for the label name.

4-Byte Area Allocation

.BLKL

Format: [Δ][<label name:>Δ].BLKLΔ<operand>

Description Reserves as many bytes of RAM area as specified by the operand in 4-byte units.

An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Examples:

```
symbol    .EQU       1
          .SECTION   area,DATA
work1:    .BLKL      1
work2:    .BLKL      symbol
          .BLKL      symbol+1
```

Notes: Be sure that this directive command is written in a section of DATA type.

Be sure to write a colon (:) for the label name.

4-Byte Area Allocation

.BLKF

Format: [Δ][<label name:>Δ].BLKFΔ<operand>

Description Reserves as many bytes of RAM area as specified by the operand in 4-byte units.

An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Examples:

```
symbol   .EQU       1
         .SECTION   area,DATA
work1:   .BLKF      1
work2:   .BLKF      symbol
         .BLKF      symbol+1
```

Notes: Be sure that this directive command is written in a section of DATA type.

Be sure to write a colon (:) for the label name.

8-Byte Area Allocation

.BLKD

Format: [Δ][<label name:>Δ].BLKDΔ<operand>

Description Reserves as many bytes of RAM area as specified by the operand in 8-byte units.

An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Examples:

```
symbol   .EQU       1
         .SECTION   area,DATA
work1:   .BLKD      1
work2:   .BLKD      symbol
         .BLKD      symbol+1
```

Notes: Be sure that this directive command is written in a section of DATA type.

Be sure to write a colon (:) for the label name.

1-Byte Data Storing

.BYTE

Format: [Δ][<label name:>Δ].BYTEΔ<operand>

Description Stores the 1-byte long data specified by the operand in ROM.

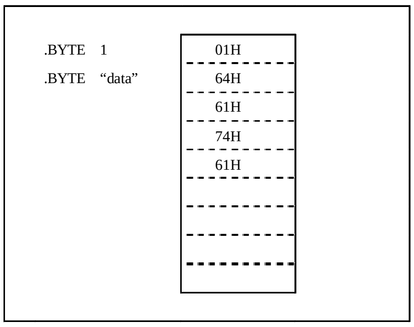
An expression or symbol can be written in the operand.

To write multiple operands, use a comma (,) to separate each operand written.

A character or string enclosed in single-quotes (') or double-quotes (") can be written in the operand. In this case, the stored data consists of ASCII code representing the characters.

Examples:

```
.SECTION    value,ROMDATA
.BYTE       1
.BYTE       "data"
.BYTE       symbol
.BYTE       symbol+1
.BYTE       1,2,3,4,5
.END
```



Notes: Be sure that this directive command is written in other than a DATA type section.

Be sure to write a colon (:) for the label name.

(signed) 2-Byte Data Storing

.WORD(S)

Format: [Δ][<label name:>Δ].WORDΔ<operand>

[Δ][<label name:>Δ].WORDSΔ<operand>

Description Stores the 2-byte long data specified by the operand in ROM.

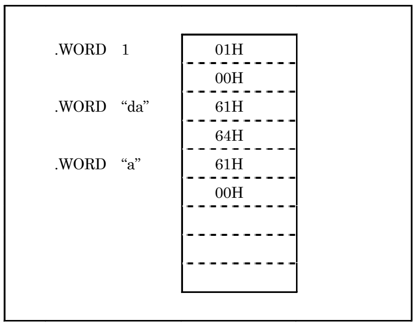
An expression or symbol can be written in the operand.

To write multiple operands, use a comma (,) to separate each operand written.

A character or string enclosed in single-quotes (') or double-quotes (") can be written in the operand. In this case, the stored data consists of ASCII code representing the characters.

Examples:

```
.SECTION    value,ROMDATA
.WORD       1
.WORD       "da","a"
.WORD       symbol
.WORD       symbol+1
.WORD       1,2,3,4,5
.END
```



Notes: Be sure that this directive command is written in other than a DATA type section.

Be sure to write a colon (:) for the label name.

A string in length of up to 2 characters can be written in the operand.

The value writable in the operand of ".WORD" is from -32,768 to 65,535, and the value writable in the operand of ".WORDS" is from -32,768 to 32,767.

3-Byte Data Storing

.ADDR

Format: [Δ][<label name:>Δ].ADDRΔ<operand>

Description Stores the 3-byte long data specified by the operand in ROM.

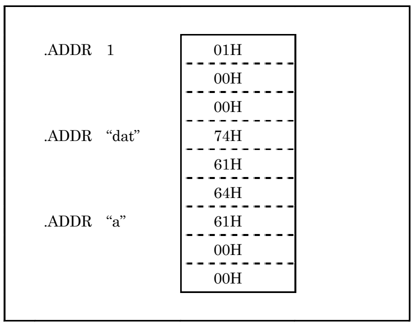
An expression or symbol can be written in the operand.

To write multiple operands, use a comma (,) to separate each operand written.

A character or string enclosed in single-quotes (') or double-quotes (") can be written in the operand. In this case, the stored data consists of ASCII code representing the characters.

Examples:

```
.SECTION    value,ROMDATA
.ADDR       1
.ADDR       "dat","a"
.ADDR       symbol
.ADDR       symbol+1
.ADDR       1,2,3,4,5
.END
```



Notes: Be sure that this directive command is written in other than a DATA type section.

Be sure to write a colon (:) for the label name.

A string in length of up to 3 characters can be written in the operand.

4-Byte Data Storing

.LWORD

Format: [Δ][<label name:>Δ].LWORDΔ<operand>

Description Stores the 4-byte long data specified by the operand in ROM.

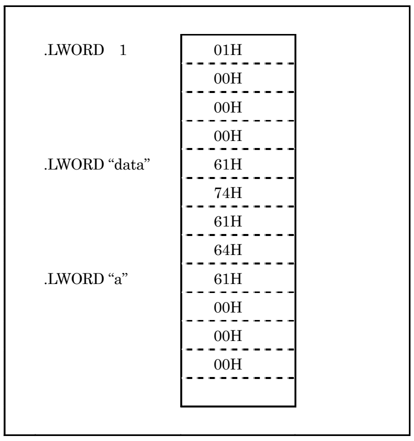
An expression or symbol can be written in the operand.

To write multiple operands, use a comma (,) to separate each operand written.

A character or string enclosed in single-quotes (') or double-quotes (") can be written in the operand. In this case, the stored data consists of ASCII code representing the characters.

Examples:

```
.SECTION    value,ROMDATA
.LWORD      1
.LWORD      "data","a"
.LWORD      symbol
.LWORD      symbol+1
.LWORD      1,2,3,4,5
.END
```



Notes: Be sure that this directive command is written in other than a DATA type section.

Be sure to write a colon (:) for the label name.

A string in length of up to 4 characters can be written in the operand.

4-Byte Data Storing

.FLOAT

Format: [Δ][<label name:>Δ].FLOATΔ<operand>

Description Stores the 4-byte long data specified by the operand in ROM.

Examples:

```
        .FLOAT    5E2
const:  .FLOAT    5e2
```

Notes: Refer to "3.5.2 Rules for Writing an Operand" for details on how to write a floating point number in the operand.

Be sure to write a colon (:) for the label name.

8-Byte Data Storing

.DOUBLE

Format: [Δ][<label name:>Δ].DOUBLEΔ<operand>

Description Stores the 8-byte long data specified by the operand in ROM.

Examples:

```
        .DOUBLE     5E2
const:  .DOUBLE     5e2
```

Notes: Refer to "3.5.2 Rules for Writing an Operand" for details on how to write a floating point number in the operand.

Be sure to write a colon (:) for the label name.

Corrects odd addresses to even addresses

.ALIGN

Format: [Δ].ALIGN

Description Corrects the address at which code in the line next to the one where this directive command is written will be stored, by changing it to an even address.

If the section type is CODE or ROMDATA, blank spaces resulting from the address correction are filled with NOP code (04H).

If the section type is DATA, the address value is incremented by 1.

If the address for which this directive command is written is an even address, no correction is made.

Examples:

```
.SECTION    program,CODE,ALIGN
MOV.W       #0,R0
.ALIGN
MOV.W       #0,R1

.SECTION    program,CODE
.ORG        0f000H
MOV.W       #0,R0
.ALIGN
MOV.W       #0,R1
.END
```

Notes: For relative sections, write ",ALIGN" in the section definition directive command ".SECTION".

9.2 Assemble Control Directive Commands

These directive commands do not generate data corresponding to themselves but controls generation of machine code for instructions. They do not modify addresses.

Table 9.2 Assemble control directive commands

Directive	**Function**
.EQU	Defines symbol.
.BTEQU	Defines bit symbol.
.END	Specifies the end of an assembly-language file.
.SB	Assigns temporary SB register value.
.SBSYM	Selects SB relative displacement addressing mode.
.SBBIT	Selects SB relative displacement addressing mode for bit symbol.
.FB	Assigns temporary FB register value.
.FBSYM	Selects FB relative displacement addressing mode.
.INCLUDE	Inserts the contents of the specified file to the location where this directive is written.
.SB_AUTO[_xxx]	Automatic Generation of SB Relative Addressing.

Numeric Value Symbol Definition

.EQU

Format: [Δ]<name>Δ.EQUΔ<operand >

Description: For the symbol, define a 32-bit signed integer value in the range (-2147483648 to 2147483647).

An expression or a symbol can be written for the operand. However, the operand value must be the one that is definite at assemble execution time.

Symbols can be specified as global.

Examples:

```
symbol      .EQU     1
symbol1     .EQU     symbol+symbol
symbol2     .EQU     2
```

Notes: symbol names cannot be entered that are forward referenced.

Bit Symbol Definition

.BTEQU

Format: [Δ]<bit symbol>Δ.BTEQUΔ<bit position>,<address value>

[Δ]<bit symbol>Δ.BTEQUΔ<bit symbol>

Description: This command defines a bit position and memory address. The symbol defined by this directive command is called a bit symbol.

By defining a bit symbol with this directive command you can write a bit symbol in the operand of a 1-bit operating instruction.

The defined bit position is a bit whose position is offset from the LSB of a specified address value of memory by a value that indicates the bit position.

An integer in the range of 0 to 65535 can be written to indicate the bit position.

An expression or a symbol can be written for the bit position. However, the operand value must be the one that is definite at assemble execution time.

An expression or a symbol can be written for the address value.

The bit symbol name can be globally specified.

Examples:

```
.GLB    flag1
one     .EQU        1
bit0    .BTEQU      0,0
bit1    .BTEQU      1,flag
bit2    .BTEQU      2,flag+1
bit3    .BTEQU      one+one,flag
bit4    .BTEQU      one,flag1
bit5    .BTEQU      bit0
```

Notes: No bit symbols can be externally referenced (written in the operand of directive command ".BTGLB") that are defined by a symbol that is indeterminate when assembled.

A bit symbol name in the operand cannot be forward referenced. Also, for the operand bit symbol, be sure to write a bit symbol name whose value is fixed when assembled.

Assembly-Language File End Declaration

.END

Format: [Δ].END

Description: This command declares the end of the source program.

The assembler only outputs the contents written in the subsequent lines after this directive command to a list file and does not perform code generation and other processing.

Examples:

```
.SECTION   tbl,romdata
.BYTE      1,2,3,4,5
.END
```

Remarks: There must always be at least one of this directive command in one assembly source file.

Notes: The as30 assembler does not detect errors in the subsequent lines after this directive command either.

Assigns temporary SB register value

.SB

Format: [Δ].SBΔ<operand>

Description: Assumes an SB register value.

The value of the SB register is assumed to be the one defined by this directive command at assemble execution time, and code is generated based on that thereafter.

A label name specified by the directive command ".SBSYM" can be used in places following this directive command line.

For instructions using a label name specified by the directive command ".SBSYM", code is generated with SB relative addressing mode, referenced as base point to the value assumed by ".SB".

An integral value in the range 0 to 0FFFFH can be written in the operand.

An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Examples:

```
.SB     80H
LDC     #80H,SB
```

Notes: This directive command only directs the assembler to assume an SB register value and does not set a value for the actual SB register value. To actually set an SB register value, write the following instruction immediately before or after this directive command.

```
Example)        LDC   #80H,SB
```

Selects SB relative displacement addressing mode

.SBSYM

Format: [Δ].SBSYMΔ<name>[,<name>...]

Description: Selects SB relative addressing mode for the name specified in the operand of this directive command.

It is for an expression of absolute 16-bit addressing mode including the name specified in the operand of this directive command that SB relative addressing mode is selected.

A name that has a relocatable value can be specified in the operand.

Examples:

```
.SB         80H
LDC         #80H,SB
.SBSYM      sym1,sym2
```

Notes: Before writing this directive command, be sure to set an SB register value with the directive command ".SB".

SB relative addressing mode is not selected for the symbols defined by the directive command ".EQU" using the label name specified by this directive command.

When writing this directive command, make sure the symbol specified by it does not duplicate other symbols specified by the directive command ".FBSYM".

In the example below, SB relative addressing mode is not selected for sym2.

```
.SBSYM      sym1
sym2        .EQU      sym1+1
```

Selects SB relative displacement addressing mode(bit mnemonics)

.SBBIT

Format: [Δ].SBBITΔ<operand>[,<operand> …]

Description: Selects SB relative addressing mode for the name specified in the operand of this directive command.

If the bit manipulating instruction has a short form, 11-bit SB relative addressing mode or 16-bit SB relative addressing mode is selected.

If the bit manipulating instruction does not have a short form, 8-bit SB relative addressing mode or 16-bit SB relative addressing mode is selected.

If the bit symbol of the operand is externally referenced, 16-bit SB relative addressing mode is selected. However, if this directive command is written specifying a short form (:S) for any mnemonic that has a short form, 11-bit SB relative addressing mode is selected.

A bit symbol defined by the directive command ".BTEQU" or ".BTGLB" can be written in the operand.

Examples:

```
.BTGLB   extbit
.SB      80H
LDC      #80H,SB
.SBBIT   bsym,extbit
BCLR     bsym       ; Select 11 bits SB
BAND     bsym       ; Select 16 bits SB
BSET     extbit     ; 16 bits SB
BSET:S   extbit     ; 11 bits SB
```

Notes: A forward referenced bit symbol can be written in the operand.

Before writing this directive command, be sure to set an SB register value with the directive command ".SB".

Assigns temporary FB register value

.FB

Format: [Δ].FBΔ<operand>

Description: Assumes an FB register value.

The value of the FB register is assumed to be the one defined by this directive command at assemble execution time, and code is generated based on that thereafter.

A label name specified by the directive command ".FBSYM" can be used in places following this directive command line.

For instructions using a label name specified by the directive command ".FBSYM", code is generated with FB relative addressing mode, referenced as base point to the value assumed by ".FB".

An integral value in the range 0 to 0FFFFH can be written in the operand.

An expression or symbol can be written in the operand. However, the operand value must be the one that is determinate at assemble execution time.

Examples:

```
.FB       80H
LDC       #80H,FB
```

Notes: This directive command only directs the assembler to assume an FB register value and does not set a value for the actual FB register value. To actually set an FB register value, write the following instruction immediately before or after this directive command.

```
Example)        LDC   #80H,FB
```

Selects FB relative displacement addressing mode

.FBSYM

Format: [Δ].FBSYMΔ<name>[,<name>...]

Description: Selects FB relative addressing mode for the name specified in the operand of this directive command.

It is for an operand of absolute 16-bit addressing mode including the name specified in the operand of this directive command that FB relative addressing mode is selected.

A name that has a relocatable value can be specified in the operand.

Examples:

```
.FB       80H
LDC       #80H,FB
.FBSYM    sym1,sym2
```

Notes: Before writing this directive command, be sure to set an FB register value with the directive command ".FB".

When writing this directive command, make sure the symbol specified by it does not duplicate other symbols specified by the directive command ".SBSYM".

Reads file into specified position

.INCLUDE

Format: [Δ].INCLUDEΔ<file name>

Description: Loads the entire content of other files into source program lines.

The contents of files loaded by this directive command are processed as an integral part of the source file into which they are loaded, as if they had been written in it.

Include files can be nested up to 9 levels.

If the include file name has an absolute path written in it, the specified directory is searched for files. If no files are found, an error results.

If the include file name has no absolute paths specified, files are searched in the order described below.

1. If the file name specified on the command line of as30 at startup time has no directories specified, the file name specified by an include directive command is searched. If the file name specified on the command line of as30 at startup time has any directory specified, the directory name specified on the command line is added to the file name specified by an include directive command, by which files are searched.
2. The directory specified by the command option -I is searched.
3. The directory set in the environment variable INC30 is searched.

Examples:

```
.INCLUDE    initial.a30
.INCLUDE    ..FILE@.inc
```

Remarks: Be sure that the file name written in the operand includes a file extension.

A string including the directive command "..FILE" or a "@" can be written in the operand.

Notes: Be careful not to include an include file in itself.

Automatic Generation of SB Relative Addressing

.SB_AUTO[_xxx]

Format:

```
[Δ].SB_AUTO
[Δ].SB_AUTO_S          ; C language function name, assembler function name
[Δ].SB_AUTO_SBVAL      ; SB register set value
[Δ].SB_AUTO_SBSYM      ; SB relative addressing target symbol
[Δ].SB_AUTO_R
[Δ].SB_AUTO_E
```

Description: The assembler selects the SB relative addressing mode.

Generates the instructions to save/restore SB register and to set register values.

Table 9.3 directive commands SB_AUTO

Directive	Function
.SB_AUTO	Shows that automatic generation of SB relative addressing will start.
.SB_AUTO_S	Shows the beginning of the function.
.SB_AUTO_SBVAL	Generates the instruction to save SB register (PUSHC) and the instruction to set register values (LDC).
.SB_AUTO_SBSYM	Selects the SB relative addressing mode for the name specified in the operand.
.SB_AUTO_R	Generates the instruction to restore SB register (POPC).
.SB_AUTO_E	Shows the end of the function.

Examples:

```
        .glb    _func1
_func1:
        .sb_auto_s          func1,_func1
        .sb_auto_sbval      _i1
        .sb_auto_sbsym      _i1,_i2,_i3
        ;
        .sb_auto_r
        rts
        .sb_auto_e
```

Notes: These directive commands are C complier-only directives, so that they cannot be written in user programs.

Depending on condition, no instructions will be generated by ".SB_AUTO_SBVAL" and ".SB_AUTO_R".

9.3 Link Control Directive Commands

These directive commands are provided for executing relocatable assembly of programs where a program is written separately in multiple files.

Table 9.4 Link directive commands

Directive	Function
.SECTION	Defines a section that is the minimum unit in which units addresses are relocated. Section information includes a section name, section type, and section attribute.
.GLB	Declares that the symbol is an external symbol. If a definition of the declared symbol is found in the same file, it is handled as an external symbol. If not, it is handled as an externally referenced symbol.
.BTGLB	Declares that the bit symbol is an external symbol.
.RVECTOR	Sets a software interrupt number and software interrupt name.
.SVECTOR	Sets a special page number and special page name.
.INITSCT	Defines a section name temporarily. This directive command is generated by an initialize function of the C startup (initsct.c). It is used exclusively by the compiler.

Section Definition

.SECTION

Format: [Δ].SECTIONΔ<section name>[,<section type>][,ALIGN]

Description: Declares the beginning of a section. The scope of a section is from here to the next section directive command or the directive command ".END".

For the section type, write CODE, ROMDATA, or DATA. If omitted, the section is handled as CODE type.

If ",ALIGN" is specified, the section is assumed to be a relative section and the start addresses of those sections in the file are aligned on even-address boundaries by the linkage editor.

If the ".ORG" directive command is written in the line next to the section directive command, the section is handled as having the absolute attribute.

Examples:

```
.SECTION        program, CODE
        NOP
.SECTION        ram, DATA
        .BLKB   10
.SECTION        dname, ROMDATA
        .BYTE   "abcd"
.END
```

Remarks: The section type and ",ALIGN" can be written in any order.

Notes: If multiple section definitions with the same section name are declared in a file, they are concatenated into one section. In that case, the word ",ALIGN" written in the second and subsequent section definitions are ignored.

Specifies global label

.GLB

Format: [Δ].GLBΔ<name>[,<name>...]

Description: Declares the label or symbol specified by a name as having the global attribute.

If the specified name is defined in the file, the label or symbol can be referenced from external files.

If not, it is assumed that the specified name is defined in an external file.

Examples:

```
.GLB        name1,name2,name3
.GLB        name4
.SECTION    program
MOV.W       #0,name1
```

Selects FB relative displacement addressing mode

.BTGLB

Format: [Δ].BTGLBΔ<bit symbol name>[,<bit symbol name>...]

Description: Declares the label or symbol specified by a name as having the global attribute.

If the specified name is defined in the file, the label or symbol can be referenced from external files.

If not, it is assumed that the specified name is defined in an external file.

Examples:

```
.BTGLB     flag1,flag2,flag3
.BTGLB     flag4
.SECTION   program
BCLR       flag1
```

Notes: Bit symbols defined by a symbol whose value is indeterminate at assemble execution time cannot be specified as externally referenced.

Set the software interrupt

.RVECTOR

Format: [Δ].RVECTORΔ<software interrupt No>,<software interrupt name>

Description: By this directive command, the assembler automatically generates a variable vector table at link time. The variable vector table is generated with a section name "vector".

The entire area of a variable vector table (256 bytes) is generated regardless of whether all software interrupt numbers are set by this directive command.

For the software interrupt number, write a value in the range 0 to 63 that is determinate at assemble time.

For the software interrupt name, write a symbol or label.

When a variable vector table is automatically generated by this directive command, information on the variable vector table is output to the linkage list file(.map) generated by the linker.

Examples:

```
.rvector   21, timerA0       ; Sets timerA0 to software interrupt number 21.
```

Remarks: Blank spaces in the variable page vector table where no software interrupt numbers are set by this directive command have values set in the order given below.

1. The value set by a link option "-VECT"
2. The value of a global label "_dummy_int"
3. The value of a global label "dummy_int"
4. The remaining other blank spaces are filled with "00H"

Notes: If, after writing this directive command, a program is written in the "vector" section, this directive command results in an error. (Do not write a program in the "vector" section.)

The software interrupt numbers specified by this directive command cannot be specified by a link option "-VECTN".

Sets the special page

.SVECTOR

Format: [Δ].SVECTORΔ<special page no>,<special page name>

Description: By this directive command, the assembler automatically generates a special page vector table at link time. The special page vector table is generated with a section name "svector".

For the special page number, write a value in the range 18 to 255 that is determinate at assemble time.

For the special page name, write a symbol or label.

When a special page vector table is automatically generated by this directive command, information on the special page vector table is output to the linkage list file(.map) generated by the linker.

Examples:

```
 .svector    250,spFunc       ;Sets spFunc to special page number 250.
```

Remarks: The section name "svector" has an area reserved for it at link time, with the area ranging from special page number 18 to the largest special page number specified.

If a blank space is found in the special page vector table after being automatically generated (where no special page numbers are specified by this directive command), the blank space is filled with FFH.

Notes: Unless the "-R8C" option is specified, this directive command cannot be used.

If, after writing this directive command, a program is written in the "svector" section, this directive command results in an error. (Do not write a program in the "svector" section.)

To set a blank space for some specific part of the area beginning with special page number 18, use the directive command ".RESERVE_AREA". In the example below, parts of the area for special page numbers 18 and 19 are made a blank space.

```
 .reservw_area     0fffd8h,4
```

Defines a section name temporarily

.INITSCT

Format: [Δ].INITSCTΔ<section name>Δ<section type>,align

[Δ].INITSCTΔ<section name>Δ<section type>,noalign

Description: Provisionally defines a section name.

This is a C language startup-only directive command.

Examples:

```
.initsct      bss_NE, data, align       ; get alignment
.initsct      bss_NO, data, noalign     ; not get alignment
```

Notes: This directive command is generated by the C language startup (initsct.c) initialization function, and is usable in only a compiler.

9.4 List Directive Dommands

This directive command controls the output information and format of the source list file. It does not affect code generation.

Table 9.5 list directive commands

Directive	**Function**
.LIST	Controls outputting of line data to list file.
.PAGE	Breaks page at specified position of list file.
.FORM	Specifies number of columns and lines in 1 page of list file.

List output

.LST

Format: [Δ].LISTΔ[ON|OFF]

Description: Controls line output to the assembler list file.

Write ".LIST OFF" to stop line output. Write ".LIST ON" to restart line output.

Even while line output to list is stopped, lines in error are forcibly output to the list file.

If this directive command is not specified, all lines are output to the list file.

Examples:

```
.LIST    ON
.LIST    OFF
```

Output a list file page break

.PAGE

Format:	[Δ].PAGEΔ<"character string">
	[Δ].PAGEΔ<'character string'>
Description:	Inserts a page break in the assembler list file.
	A string specified in the operand, if any, is changed for the "SOURCE LIST" part of the header of the new page before being output.
	Enclose the operand in single-quotes (') or double-quotes (") when writing it.
	If the operand is omitted, the default string (SOURCE LIST) or the string specified by an immediately preceding .PAGE is output.

Examples:

```
.PAGE
.PAGE    "strings"
.PAGE    'strings'
```

Notes: Up to 135 characters can be output to the header. If the number of columns in the list file is specified by the directive command ".FORM", the maximum number of characters that can be output is limited to the "number of list file columns -65".

Specifies number of columns and lines

.FORM

Format: [Δ].FORMΔ[<number of lines>],[<number of columns>]

Description: This command specifies the number of lines per page of the assembler list file in the range of 20 to 255.

This command specifies the number of columns per page of the assembler list file in the range of 80 to 295.

The contents specified by this directive command become effective beginning with the page next to one where the command is written. However, if this directive command is written in the first line of the assembly source file, the specified contents become effective beginning with the first page.

If this directive command is not specified, the assembler list file is output with the number of lines = 66 and the number of columns = 200.

Examples:

```
 .FORM    20,80
 .FORM    60
 .FORM    ,100
 .FORM    line,culmn
```

Remarks: This command can be written for multiple instances in one assembly source file.

A symbol can be used to describe the number of lines and the number of columns.

An expression can be used to describe the number of lines and the number of columns.

If you specify only the number of columns in the operand, be sure to enter a comma (,) immediately before the numeric value you write for the number of columns.

Notes: Symbols cannot be used that are forward referenced.

9.5 Conditional Assembly Directive Commands

These directive commands specify whether to assemble a specified range of lines

Table 9.6 conditional assembly directive commands

Directive	**Function**
.IF	Specifies the beginning of a conditional assembly block and evaluates the condition.
.ELIF	Evaluates the second or later conditions when multiple conditional blocks are written.
.ELSE	Specifies the beginning of a block to be assembled when all conditions are false.
.ENDIF	Specifies the end of a conditional assembly block.

Conditional assemble control

.IF

Format:
```
[Δ].IFΔ<conditional expression>
[Δ] body
[Δ].ENDIF
```

Description: Indicates the beginning of a conditional assemble block.

The condition written in the operand is tested and if it evaluates to true, the body that follows is assembled.

The lines assembled when the condition is true include the directive command lines ".ELIF" and ".ELSE", the one preceding ".ENDIF".

All instructions writable in the source program of as30 can be written in a conditional assemble block.

Example of an conditional expression :

```
sym<1
sym   <   1
sym+2 < data1
sym+2 < data1+2
'smp1' == name
```

Examples:

```
.IF       TYPE = = 0
          .byte      "Proto Type Mode"
.ELIF     TYPE>0
          .byte      "Mass Production Mode"
.ELSE
          .byte      "Debug Mode"
.ENDIF
```

Rules for writing a conditional expression:

Only one conditional expression can be written in the operand of the directive command.

Be sure that a conditional operator is written in the conditional expression.

The operators listed below can be written.

Table 9.7 Conditional Operators in the .IF and .ELIF Directive Commands

Directive	Function
>	True when the left value is greater than the right value
<	True when the left value is less than the right value
>=	True when the left value is greater than or equal to the right value
<=	True when the left value is less than or equal to the right value
==	True when the left and right values are equal
!=	True when the left and right values are not equal

Operations on conditional expressions are performed in signed 32 bits.

A symbol can be written on the left and right sides of the conditional operator.

An expression can be written on the left and right sides of the conditional operator. Follow the "rules for writing a program" and "rules for writing an expression" to write an expression.

A string can be written on the left and right sides of the conditional operator. Be sure that the string written is enclosed in single-quotes (') or double-quotes ("). The relative sizes of strings are determined by the value of the character code.

"ABC" < "CBA" → 414243 < 434241; therefore, condition is true.
"C" < "A" → 43 < 41; therefore, condition is false.

A space or tab can be written before and after the conditional operator.

An expression can be written in the operands of directive commands ".IF" and ".ELIF".

Notes: Overflows and underflows resulting from operations are not tested.

Symbols cannot be forward referenced (i.e., symbols defined after this directive command line cannot be referenced). If a forward referenced symbol or an undefined symbol is written, its value is assumed to be 0 when the expression is tested. In that case, no error messages are output.

For expressions on the left and right sides of the conditional operator, set a value that is basically determinate at the assemble execution time.

Conditional assemble control

.ELIF

Format:

```
[Δ].IFΔ<conditional expression>
[Δ] body
[Δ].ELIFΔ<conditional expression>
[Δ] body
[Δ].ENDIF
```

Description: Use this command to write a condition in combination with ".IF" if you want to specify multiple conditions for conditional assemble to be performed.

The assembler resolves the condition written in the operand and, if it is true, assembles the body that follows.

If condition is true, lines are assembled up to and not including the line where directive command ".ELIF", ".ELSE" or ".ENDIF" is written.

Examples:

```
.IF      TYPE = = 0
         .byte    "Proto Type Mode"
.ELIF    TYPE>0
         .byte    "Mass Production Mode"
.ELSE
         .byte    "Debug Mode"
.ENDIF
```

Remarks: This directive command can be written for multiple instances in one conditional assemble block.

Conditional assemble control

.ELSE

Format:

```
[Δ].IFΔ<conditional expression>
[Δ]body
[Δ].ELIFΔ<conditional expression>
[Δ]body
[Δ].ELSE
[Δ]body
.ENDIF

[Δ].IFΔ<conditional expression>
[Δ]body
[Δ].ELIFΔ<conditional expression>
[Δ]body
[Δ].ELSE
[Δ]body
[Δ].ENDIF
```

Description: When all conditions are false, this command indicates the beginning of the lines to be assembled.

In this case, lines are assembled up to and not including the line where directive command ".ENDIF" is written.

Examples:

```
.IF       TYPE = = 0
          .byte     "Proto Type Mode"
.ELIF     TYPE>0
          .byte     "Mass Production Mode"
.ELSE
          .byte     "Debug Mode"
.ENDIF
```

Remarks: This directive command can be written less than once in a conditional assemble block.

This directive command does not have an operand.

Conditional assemble control

.ENDIF

Format:
```
[Δ].IFΔ<conditional expression>
[Δ] body
[Δ].ENDIF
```

Description: This command indicates the end of the conditional assemble block.

Examples:
```
.IF      TYPE = = 0
         .byte    "Proto Type Mode"
.ELIF    TYPE>0
         .byte    "Mass Production Mode"
.ELSE
         .byte    "Debug Mode"
.ENDIF
```

Remarks: Always make sure that there is at least one instance of this directive command in a conditional assemble block.

This directive command does not have an operand.

9.6 Macro Directive Commands

These directives define macro functions and repeat macro functions.

Table 9.8 macro directive commands

Directive	Function
.MACRO	Defines a macro name and the beginning of a macro body.
.ENDM	Specifies the end of a macro body.
.EXITM	Terminates macro body expansion.
.LOCAL	Declares a local label in a macro.
.MREPEAT	Specifies the beginning of a repeat macro body.
.ENDR	Specifies the end of a repeat macro body.
..MACPARA	Indicates the number of arguments in a macro call.
..MACREP	Indicates the count of repeat macro body expansions.
.LEN	Indicates the number of characters in a specified string.
.INSTR	Indicates the start position of a specified string in another specified string.
.SUBSTR	Extracts a specified number of characters from a specified position in a specified string.

Defines a macro name and the beginning of a macro body

.MACRO

Format: Macro definition

```
  [Δ]<macro name>Δ.MACROΔ[<formal parameter>[,<formal parameter>...] ]
  [Δ]   body
  [Δ].ENDM
```

Macro call

```
  [Δ]<macro name>Δ[ <actual parameter>[,<actual parameter>...] ]
```

Description: This command defines a macro name.

Follow "Rules for Writing a Program" and "Rules for Writing a Name" to write a macro name.

This command indicates the beginning of macro definition.

Example of a macro definition description:

```
mac      .MACRO      p1,p2,p3
        .IF    ..MACPARA = = 3
               .IF    'p1' = = 'byte'
                      MOV.B    #p2,p3
               .ELSE
                      MOV.W   #p2,p3
               .ENDIF
        .ELIF      ..MACPARA = = 2
               .IF    'p1' = ='byte'
                      MOV.B    p2,R0L
               .ELSE
                      MOV.W   p2,R0
               .ENDIF
        .ELSE
               MOV.W   R0,R1
        .ENDIF
.ENDM
```

Example of a macro invocation description:

```
mac   word,10,R0
```

Example of macro expansion:

```
mac     word,10,R0
          .IF          3==3
               .IF      'word' = ='byte'
               .ELSE
                        MOV.W#10,R0
               .ENDIF
          .ELIF       3==2
          .ELSE
          .ENDIF
.ENDM
```

Formal parameters:

Follow "Rules for Writing Program" and "Rules for Names" to write formal macro parameters.

Define the formal macro parameters with names different from each other including those in nested macro definitions.

Make sure the formal parameters written in the operand of the directive command ".MACRO" are written within the macro body.

Up to 80 formal parameters can be written within the range not exceeding the number of characters writable in one line.

Actual parameters:

Write actual parameters corresponding one for one to the formal parameters at the time of macro invocation.

To write a special character in an actual parameter, enclose it in double quotes.

A label, global label, or symbol can be written for the actual parameter.

An expression can be written for the actual parameter.

Expansion of actual parameters:

Formal parameters are replaced with actual parameters in the order they are written, from left to right.

If, while a formal parameter is defined, there is no actual parameter for it at macro invocation, code for the formal parameter is not output.

If the number of formal parameters is greater than that of actual parameters, no code is output for the formal parameters that do not have the corresponding actual parameters.

If formal parameters written in the body are enclosed in single quotes ('), the corresponding actual parameters are enclosed in single quotes when they are output.

If one actual parameter contains a comma (,) while at the same time it is enclosed in parentheses, conversion is performed including these parentheses.

If the number of actual parameters is greater than that of formal parameters, the actual parameters that do not have the corresponding formal parameters are not processed.

Example of actual parameter expansion:

Example of macro definition

```
name    .MACRO   string
        .BYTE    'string'
.ENDM
```

Example of macro call-1

```
name    "name,address"
        .BYTE    'name,address'
```

Example of macro call -2

```
name    (name,address)
        .BYTE    '(name,address)'
```

Notes: All character strings enclosed with double quotations indicate the character strings themselves and nothing else. Therefore, do not enclose the formal parameters with double quotations.

If the number of actual parameters does not match that of formal parameters, the as30 assembler outputs a warning message.

Specifies the end of a macro body

.ENDM

Format: [Δ]<macro name>Δ.MACRO
[Δ] body
[Δ].ENDM

Description: This command indicates that the body of one macro definition is terminated here.

Examples:

```
lda    .MACRO    value
       MOV.W     #value,A0
.ENDM

lda    0
       MOV.W     #0,A0
```

Remarks: Always make sure that this command corresponds to directive command ".MACRO" as you write it.

Terminates macro body expansion

.EXITM

Format:
```
[Δ]<macro name>Δ.MACRO
[Δ]body
[Δ].EXITM
[Δ]body
[Δ].ENDM
```

Description: This command stops expanding the macro body and transfers control to the nearest ".ENDM".

Examples:
```
data1   .MACRO      value
        .IF    value = = 0
               .EXITM
        .ELSE
               .BLKB   value
        .ENDIF
.ENDM

data1   0
        .IF    0 = = 0
               .EXITM
        .ENDIF
```

Remarks: Make sure that the command is written within the body of a macro definition.

Declares a local label in a macro

.LOCAL

Format: [Δ].LOCALΔ<label name> [,<label name>...]

Description: This command declares that the label written in the operand is a macro local label.

Macro local labels are allowed to be written for multiple instances with the same name providing that they differently macro defined or they are written outside macro definition.

Examples:

```
name   .MACRO
       .LOCAL   m1    ; 'm1' is macro local label
m1:
       nop
       jmp      m1
.ENDM
```

Remarks: Always make sure that this directive command is written within the macro body.

Make sure that macro local label declaration by this directive command is entered before you define the label name.

To write a macro local label name, follow the rules for writing name in Section 3, "Assembler Language Description Rules".

Multiple labels can be written in the operand of this directive command providing that they are separated with a comma. In this case, up to 100 labels can be entered.

Notes: If macro definitions are nested, macro local labels in the macro that is defined within macro definition are not allowed to be used in the same name again.

The maximum number of macro local labels that can be written in one assembly source file including the contents of include files is 65,535.

Specifies the beginning of a repeat macro body

.MREPEAT

Format:
```
[Δ][<label>:Δ].MREPEATΔ<numeric value>
[Δ]body
[Δ].ENDR
```

Description: This command indicates the beginning of a repeat macro.

The macro body is expanded repeatedly a specified number of times.

The maximum number of repetitions that can be specified is 65,535.

Repeat macros can be nested in up to 65,535 levels.

The macro body is expanded into the line in which this directive command is written.

Examples:

```
.MREPEAT      3
        nop
.ENDR
```

Example of expansion:

```
.MREPEAT      3
        nop
        nop
        nop
.ENDR
```

Example of a combination with macro definition:

```
rep      .MACRO      num
         .MREPEAT   num
         .IF    num > 49
                .EXITM
         .ENDIF
         nop
         .ENDR
.ENDM

rep      3
         nop
         nop
         nop
```

Remarks: A symbol can be written in the operand.

An expression can be written in the operand.

Directive command ".EXITM" can be written in the body.

Notes: Forward referenced symbols cannot be used here.

Specifies the end of a repeat macro body

.ENDR

Format: [Δ][<label>:Δ].MREPEATΔ<numeric value>
[Δ] body
[Δ].ENDR

Description: This command indicates the end of a repeat macro.

Examples:

```
.MREPEAT      3
          nop
.ENDR
```

Indicates the number of arguments in a macro call

..MACPARA

Format: [Δ]..MACPARA

Description: Indicates the number of arguments in a macro invocation.

It can be written in the body of a macro definition by ".MACRO".

This directive command can be written as a term of an expression.

Examples: The assembler checks the number of macro actual parameters as it executes conditional assemble.

```
          .GLB        mem
    name  .MACRO      f1,f2
          .IF     ..MACPARA = = 2
                  ADD.W       f1,f2
          .ELSE
                  ADD.W       R0,f1
          .ENDIF
    .ENDM

    name  mem
          .ELSE
                  ADD.W       R0,mem
          .ENDIF
```

Notes: If this command is written outside the macro body defined by ".MACRO", its value is made 0. In that case, no error messages are output.

Indicates the count of repeat macro body expansions

..MACREP

Format: [Δ]..MACREP

Description: This command indicates how many times the repeat macro is expanded.

This command can be written in the body of a macro definition defined by ".MREPEAT".

This command can be written in the conditional assemble operand.

This directive command can be written as a term of an expression.

Examples:

```
.MREPEAT           3
        MOV.W      R0,..MACREP
.ENDR

        MOV.W      R0, 1
        MOV.W      R0, 2
        MOV.W      R0, 3

.GLB    mem
mclr    .MACRO     value,name
        .MREPEAT   value
        MOV.W      #0, name +..MACREP
        .ENDR
.ENDM

mclr    3,mem
        .MREPEAT   3
        MOV.W      #0,mem+1
        MOV.W      #0,mem+2
        MOV.W      #0,mem+3
        .ENDR
```

Notes: If this command is written outside the macro body, its value is made 0. In that case, no error messages are output.

Indicates the number of characters in a specified string

.LEN

Format: [Δ].LENΔ{"<character string>"}

[Δ].LENΔ{'<character string>'}

Description: This command indicates the length of the character string that is written in the operand.

Always be sure to enclose the operand with { }.

Space or tab can be written between this directive command and the operand.

The 7-bit ASCII code characters including a space and tab can used to write a character string.

Always be sure to enclose the character string with quotations as you write it.

This directive command can be written as a term of an expression.

Examples:

```
        .byte           .LEN{"string"}
```

Example of expansion:

```
        .byte           6
```

Example1 of a combination with macro definition:

```
bufset  .MACRO   f1,f2
        buffer@f1:  .BLKB     .LEN{ 'f2' }
.ENDM

bufset  1,Printout_data
        buffer1     .BLKB     13

bufset  2,Sample
        buffer2     .BLKB     6
```

Example2 of a combination with macro definition:

```
buf     .MACRO   f1
        buffer:     .BLKB     .LEN{ "f1" }
.ENDM

buf     1,data      ; 'data' is not expanded.
        buffer      .BLKB     2
```

Notes: Kanji and other 8-bit code are not processed correctly. However, the as30 assembler does not output errors.

If you want a macro parameter to be expanded as a character string, enclose the macro name with single quotations as you write it. If enclosed with double quotations, the character string length of the formal parameter written in macro definition is assumed.

Indicates the start position of a specified string in another specified string

.INSTR

Format: [Δ].INSTRΔ{"<character string>","<search character string>",<search start position>}

[Δ].INSTRΔ{'<character string>','<search character string>',<search start position>}

Description: This command indicates a position in the character string specified in the operand at which a search character string begins.

A position can be specified at which you want the assembler to start searching a character string.

Always be sure to enclose the operand with { }.

A symbol can be written in the search start position.

If you specify 1 for the search start position, it means the beginning of the character string.

The 7-bit ASCII code characters including a space and tab can be used to write a character string.

Always be sure to enclose the character string with quotations as you write it.

This directive command can be written as a term of an expression.

Examples: This example extracts the position (7) of the character string "se" from the beginning (top) of the specified character string (japanese).

```
point_set  .MACRO     source,dest,top
           point      .EQU    .INSTR{'source','dest',top}
.ENDM

point_set  japanese,se,1
           point      .EQU    7
```

Notes: The value is rendered 0 if a search character string is longer than the character string itself. The value is rendered 0 if a search character string is not included in the character string. The value is rendered 0 if the search start position is assigned a value greater than the length of the character string.

Kanji and other 8-bit code are not processed correctly. However, the as30 assembler does not output errors.

If you want a macro argument to be expanded as a character string, enclose the parameter name with single quotations as you write it. Note that if you enclose a character string with double quotations, the character string itself is expanded.

Extracts a specified number of characters from a specified position in a specified string

.SUBSTR

Format: [Δ].SUBSTRΔ{"<CS>",<ES>,<NC>}

[Δ].SUBSTRΔ{'<CS>',<ES>,<NC>}

CS : character string
ES : extract start position
NC : number of characters to be extract

Description: This command extracts a specified number of characters from the specified position of a character string.

Always be sure to enclose the operand with { }.

Always be sure to write the character string, extract start position, and the number of characters to be extracted.

If you specify 1 for the extract start position, it means the beginning of the character string.

The 7-bit ASCII code characters including a space and tab can be used to write a character string.

Always be sure to enclose the character string with quotations as you write it.

Examples: The length of the character string that is given as actual parameter of the macro is given to the operand of ".MREPEAT".

".MACREP" is incremented 1 -> 2 -> 3 -> 4 each time the ".BYTE" line is executed. Consequently, the character string that is given as actual parameter of the macro is given successively to the operand of ".BYTE" one character at a time beginning with the first character in that character string.

```
name  .MACRO     data
      .MREPEAT   .LEN{ 'data' }
      .BYTE      .SUBSTR{ 'data',..MACREP,1 }
      .ENDR
.ENDM

name  ABCD
      .BYTE      "A"
      .BYTE      "B"
      .BYTE      "C"
      .BYTE      "D"
```

Notes: The value is rendered 0 if the extract start position is assigned a value greater than the length of the character string itself. The value is rendered 0 if the number of characters to be extracted is greater than the length of the character string itself. The value is rendered 0 if you specify 0 for the number of characters to be extracted.

Kanji and other 8-bit code are not processed correctly. However, the as30 assembler does not output errors.

If you want a macro argument to be expanded as a character string, enclose the parameter name with single quotations as you write it. Note that if you enclose a character string with double quotations, the character string itself is expanded.

9.7 Inspector Information Directive Commands

This directive command controls the output inspector information.

Table 9.9 inspector information directive commands

Directive	Function
.INSF	Specifies the start of a function (subroutine) in inspector information.
.EINSF	Specifies the end of a function (subroutine) in inspector information.
.CALL,.CALLIND	Specifies where to call a function (subroutine) in inspector information.
.STK	Specifies a stack in inspector information.

Define start of function of inspector information

.INSF

Format: [Δ].INSFΔ<Function (subroutine) start label>,<storage class>,<frame size>

Description: Defines start-of-function (subroutine) information of the inspector information.

Define a range from the start-of-function (subroutine) information to the directive command ".EINSF" as one function (subroutine) information.

Examples:

```
.INSF     glbfunc,G,0
glbfunc:
      ;
.EINSF

.INSF     locfunc,S,0
locfunc:
      ;
.EINSF
```

Remarks: For the storage class, write either "G (global label)" or "S (local label)".

For the frame size, write an integral value.

Notes: When this directive command is written, be sure that the directive command ".EINSF" is written.

This directive command is provided for exclusive use in assembly language description, so that if this directive command is written in the asm functions of NC30, an error results.

This directive command is valid when a command option "-finfo" is specified.

For the start-of-function (subroutine) label name, be sure that a label written for it is defined in the assembler file.

Define end of function of inspector information

.EINSF

Format: [Δ].EINSF

Description: Defines end-of-function (subroutine) information of the inspector information.

Define a range from ".INSF" to the end-of-function (subroutine) information as one function (subroutine) information.

Examples:

```
        .INSF      glbfunc,G,0
glbfunc:
                   ;
        .EINSF
```

Notes: When this directive command is written, be sure that the directive command ".INSF" is written.

This directive command is provided for exclusive use in assembly language description, so that if this directive command is written in the asm functions of NC30, an error results.

This directive command is valid when a command option "-finfo" is specified.

Define called function of inspector information

.CALL / .CALLIND

Format: [Δ].CALLΔ<called function (subroutine) name>,<storage class>

[Δ].CALLIND

Description: Defines called function (subroutine) information of the inspector information.

Examples:

```
.INSF    glbfunc,G,0
                ;
         .CALL  glbsub,G
         jsr    glbsub
                ;
         .CALL  locsub,S
         jsr    locsub
                ;
         .CALLIND
         jsri.w    extFunc
                ;
.EINSF
```

Remarks: The information on called function (subroutine) set by this directive command is output to the stack usage information file generated by the optimizing linkage editor, wherein it becomes referenced by CallWalker.

For the storage class, write either "G (global label)" or "S (local label)".

Determine how to define ".CALL" or ".CALLIND" by a jump instruction or subroutine call instruction written immediately following it.

Directive	**Function call**	**Symbol classification in CallWalker**
CALL	jmp, jsr, jmps, jsrs	Displayed as C/C++ function
.CALLIND	jmpi, jsri	Address reference unresolved function

Notes: Write this directive command within the range of the start-of-function and end-of-function information of the inspector information.

This directive command is valid when a command option "-finfo" is specified

Be sure that this directive command is written immediately before a jump instruction or subroutine call instruction.

For the called function (subroutine) of ".CALL", be sure to write a defined or referenced symbol (label).

This directive command only sets called function (subroutine) information. Therefore, use the frame size of the directive command ".INSF" or the stack size of the directive command ".STK" to set a stack size needed for the function (subroutine) call.

Define stack information of inspector information

.STK

Format: [Δ].STKΔ<stack size>

Description: Defines stack information of the inspector information.

Examples:

```
        .INSF       glbfunc, G, 0
glbfunc:
        ;
        .STK        2          ;2 byte push
        PUSH.W      R0
        ;
        .STK        -2         ;2 byte pop
        POP.W       R0
        ;
        .EINSF
```

Remarks: For the stack size, write an integral value.

Notes: Write this directive command within the range of the start-of-function and end-of-function information of the inspector information.

This directive command is valid when a command option "-finfo" is specified

9.8 Extended Feature Directive Commands

The extended feature directive commands consist of those that affect code generation and those that do not.

Table 9.10 Extended Feature Directive Commands that Affect Code Generation

Directive	Function
.ID	Set ID code.
.OFSREG	Set option function select register value. (Used for the R8C family)
.PROTECT	Set ROM code protect value. (Used for the M16C series)
.RESERVE_AREA	Handle a set area as reserved.

Table 9.11 Extended Feature Directive Commands that Do Not Affect Code Generation

Directive	Function
.ASSERT	Output strings written in operands to standard error output device or file.
?	Specify temporary label definitions and references.
..FILE	Indicate the assembler source file name being processed by as30.
@	Concatenate strings before and after @ to handle them as a single string.
.DEFINE	Defines string symbol.

Set ID code of ID code check feature

.ID

Format: [Δ].IDΔ"<ID code character string>"

[Δ].IDΔ"#<ID code numeric value>"

Description: Stores the specified ID code as 8-bit data at the ID code store address listed below.

Option	ID code store address
Not set	FFFDFH, FFFE3H, FFFEBH, FFFEFH, FFFF3H, FFFF7H, FFFFBH
-R8C/-R8CE	FFDFH, FFE3H, FFEBH, FFEFH, FFF3H, FFF7H, FFFBH

Use a string or value to specify ID code. To use a string, specify in up to 7 characters because ID code is converted to ASCII code before being stored.

To use a value, add a "#" at the beginning and specify a value in up to 14 digits using a hexadecimal number.

The specified ID code is output to the absolute file (.abs), Motorola S format file (.mot), Intel HEX format file (.hex), and binary format file (.bin). Since it also is output to the linkage list file (.map) and ID file (.id), check these files to confirm the set value.

If the directive command ".OFSREG" (or ".PROTECT") is not written while the directive command ".ID" is written, the assembler assumes that ".OFSREG 0FFH" (or ".PROTECT 0FFH") is specified. For this reason, the set value of the option function select register (or ROM code protect control address) becomes as follows:

Directive command ".ID"	Directive command ".OFSREG" (or ".PROTECT")	Set value
Specified	Specified	The value set by directive command ".OFSREG" (or ".PROTECT")
Specified	Not specified	0FFH
Not specified	Specified	The value set by directive command ".OFSREG" (or ".PROTECT")
Not specified	Not specified	The value written in the source program

Example where a numeric value is used:

a) When a 14-digit value is set for ID code

```
.id    ″#11223344556677″
```

ID code store address	0FFDFH	0FFE3H	0FFEBH	0FFEFH
ID code	11H	22H	33H	44H

ID code store address	0FFF3H	0FFF7H	0FFFBH
ID code	55H	66H	77H

b) When a value of less than 14 digits is set for ID code

```
.id    ″#1234567″
       ; ID    code 12H, 34H, 56H, 70H, 00H, 00H, 00H is set
```

c) When a value for ID code is omitted

```
.id    ″#″
       ; ID code 00H, 00H, 00H, 00H, 00H, 00H, 00H is set
```

Example where a string is used:

a) When a 7-character string is set for ID code

```
.id     ″smpcode″
```

ID code store address	0FFDFH	0FFE3H	0FFEBH	0FFEFH
ID code	73H	6DH	70H	63H

ID code store address	0FFF3H	0FFF7H	0FFFBH
ID code	6FH	64H	65H

b) When a string of less than 7 characters is set for ID code

```
.id     ″smp″
        ; ID code 73H, 6DH, 70H, 00H, 00H, 00H, 00H is set
```

c) When a string for ID code is omitted

```
.id     ″″
        ; ID code FFH, FFH, FFH, FFH, FFH, FFH, FFH is set
```

Notes: For details about the ID code check feature, see the hardware manual for the MCU used.

If this directive command is written in multiple assembler source files, a warning will result in optlnk. (The value of the first object file linked is assumed.)

To set this directive command and the directive command ".OFSREG" (or ".PROTECT"), be sure that they are set in the same source file.

Set value in option function select register

.OFSREG

Format: [Δ].OFSREGΔ<numeric value>

Description: Stores the specified value in the R8C family's option function select register (address 0FFFFH). Specify a value in the range 0 to 0FFH. To use this directive command, specify the -R8C or -R8CE option.

The specified value is output to the absolute file (.abs), Motorola S format file (.mot), Intel HEX format file (.hex), and binary format file (.bin). Since it also is output to the linkage list file (.map) and ID file (.id), check these files to confirm the set value.

Examples:

```
; fixed vector section
;--------------------------------------------------------------
        .org    0FFFCh
RESET:
        .lword    start
        .ofsreg   0FFH     ; Sets 0FFH in the option function select register.
```

Notes: For details about the option function select register, see the hardware manual for the MCU used.

If this directive command is written in multiple assembler source files, a warning will result in optlnk. (The value of the first object file linked is assumed.)

If neither the -R8C nor the -R8CE option is specified, assembler processes this directive command as the directive command ".PROTECT".

To set this directive command and the directive command ".ID", be sure that they are set in the same source file.

If this directive command is not written while the directive command ".ID" is written, the assembler assumes 0FFH as the set value of the option function select register.

Set value at ROM code protect control address

.PROTECT

Format: [Δ].PROTECTΔ<numeric value>

Description: Stores the specified value at the M16C series' ROM code protect control address (address 0FFFFFH). Specify a value in the range 0 to 0FFH. To use this directive command, do not specify the -R8C or -R8CE option.

The specified value is output to the absolute file (.abs), Motorola S format file (.mot), Intel HEX format file (.hex), and binary format file (.bin). Since it also is output to the linkage list file (.map) and ID file (.id), check these files to confirm the set value.

Examples:

```
; fixed vector section
;----------------------------------------------------------------
        .org   0FFFCh
RESET:
        .lword    start
        .protect   0FFH   ; Sets 0FFH at ROM code protect control address.
```

Notes: For details about the ROM code protect feature, see the hardware manual for the MCU used.

If this directive command is written in multiple assembler source files, a warning will result in optlnk. (The value of the first object file linked is assumed.)

If the -R8C or -R8CE option is specified, the assembler processes this directive command as the directive command ".OFSREG".

To set this directive command and the directive command ".ID", be sure that they are set in the same source file.

If this directive command is not written while the directive command ".ID" is written, the assembler assumes 0FFH as the set value at the ROM code protect control address.

Set reserved area

.RESERVE_AREA

Format: [Δ].RESERVE_AREAΔ<reservation area beginning address>,<reservation area size>

Description: Sets an area of the specified size beginning with the specified start address as a reserved area.

No programs can be located in the reserved area.

The values defined in the operands must be those that are determinate at assemble execution time.

The values that can be set in the operands are confined within the maximum address of the CPU used.

Examples:

```
 .RESERVE_AREA   0fffd8h,4
      ; Sets 4 bytes from the address 0fffd8h as a reserved area.
```

Specified String Output

.ASSERT

Format: [Δ].ASSERTΔ"<character string>"

[Δ].ASSERTΔ"<character string>" > <file name>

[Δ].ASSERTΔ"<character string>" >> <file name>

Description: This command outputs a character string written in the operand to a standard error output device when assembling the source program.

If a file name is specified, the character string written in the operand is output to the file.

With an absolute path given to the file name, as30 generates the file in the given directory.

With no absolute path given to the file name,

1. In an instance in which no directory is designated for the file name designated in the command line at the time of starting up as30:
as30 generates the file specified by this command in the current directory.

2. In an instance in which a directory is designated for a file name designated in the command line at the time of starting up AS30:
as30 generates the file with the directory of the file designated in the command line.

If ..FILE command is specified as a file name, as30 generates the file in same directory as the directory of the file designated in the command line at the time of starting up as30.

Examples: Message is output to file sample.dat.

```
    .ASSERT     "string" > sample.dat
```

Message is added to file sample.dat.

```
    .ASSERT     "string" >> sample.dat
```

Message is output to a file bearing the same name as the currently processed file except the extension.

```
    .ASSERT     "string" > ..FILE
```

Remarks: Always be sure to insert space or tab between the directive command and the operand.

Always be sure to enclose the character string in the operand with double quotations.

If you want the character string to be output to a file, specify the file name after ">" or ">>".

The symbol > directs the assembler to create a new file and output a message to that file. If there is an existing file of the same name, that file is overwritten.

The symbol >> directs the message is added to the contents of the specified file. If the specified file cannot be found, the assembler creates a new file in that name.

Space or tab can be inserted before and after ">" or ">>".

Directive command "..FILE" can be written in the file name.

Defines and references a temporary label

?

Format: [Δ]?:
<mnemonic>Δ?+
<mnemonic>Δ?-

Description: This command defines a temporary label.

The assembler references a temporary label that is defined immediately before or after an instruction.

A temporary file can be defined and referenced within the same file.

Up to 65,535 temporary files can be defined in a file. In this case, if .INCLUDE is written in the file, the maximum number of temporary files you can enter (= 65,535) includes those in the include file.

The temporary labels generated by the assembler are output to a list file.

The temporary labels are changed into tl0001,tl0002 ... and tlFFFF.

Examples:

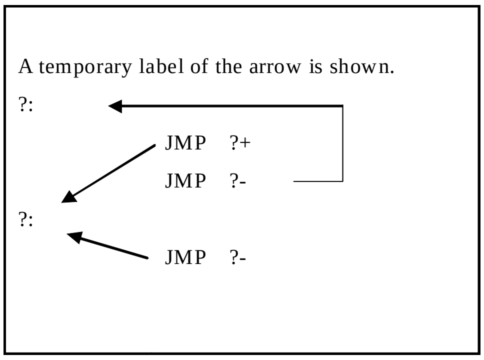
```
?:
        JMP     ?+
        JMP     ?-
?:
        JMP     ?-
```

Remarks: Write "?:" in the line where you want it to be defined as a temporary label.

If you want to reference a temporary label that is defined immediately before an instruction, write "?-" in the instruction operand.

If you want to reference a temporary label that is defined immediately after an instruction, write "?+" in the instruction operand.

Notes: The labels that can be referenced are only the label defined before or after an instruction.



Indicates the assembly source file name

..FILE

Format: [Δ]..FILE

Description: This command expands a file name into the one that is being processed by as30 (assembly source file or include file).

Examples: <sample.a30>

```
.INCLUDE     incfile.inc
.INCLUDE     ..FILE@.inc                 ..... (1)
.ASSERT      "comment" > ..FILE          ..... (2)
```

<incfile.inc>

```
.INCLUDE     ..FILE                      ….. (3)
.ASSERT      "comment" > ..FILE@.mes     ….. (4)
```

In the case of above example, they are expanded as follows.

(1) .INCLUDE sample.inc
(2) .ASSERT "comment" > sample
(3) .INCLUDE incfile
(4) .ASSERT "comment" > incfile.mes

In the case of above example, if command option -F is specified, the character strings of "..FILE" of (3) and (4) are changed to "sample" not "incfile".

(1) .INCLUDE sample.inc
(2) .ASSERT "comment" > sample
(3) .INCLUDE sample
(4) .ASSERT "comment" > sample.mes

Remarks: This command can be written in the operands of directive commands ".ASSERT" and ".INCLUDE".

Notes: The file name that can be read in by this directive command is a file name with its extension and path excluded.

If command option -F is specified, "..FILE" is fixed to an assembly source file name that is specified in the command line. If this option is not specified, the command denotes the file name where "..FILE" is written.

Concatenates character strings

@

Format: <character string>@<character string>

<character string>@<character string>[@<character string>...]

Description: This command concatenates macro arguments, macro variables, reserved symbols, expanded file name of directive command "..FILE", and specified character strings.

Examples: If the currently processed file name is "sample1.a30", a message is output to the sample.dat file.

```
    .ASSERT      "sample" > ..FILE@.dat
```

A macro definition like the one shown below can be entered:

```
  mov_nibble      .MACRO    p1,src,p2,dest
          MOV@p1@p2    src,dest
  .ENDM

  mov_nibble      .L,R0L,H,[A0]
          MOVLH          R0L,[A0]
```

Remarks: Spaces and tabs entered before and after this directive command are concatenated as a character string.

A character string can be written before and after this directive command.

When you use @ for character data (40H), be sure to enclose @ with double quotations ("). When a string including @ is enclosed with single quotation, strings before and after @ are concatenated.

This command can be written for multiple instances in one line.

Notes: If you want a concatenated character string to be a name, do not insert spaces and tabs before and after this directive command.

Defines a replacement symbol

.DEFINE

Format: [Δ]<symbol name>Δ.DEFINEΔ<character string>

[Δ]<symbol name>Δ.DEFINEΔ'<character string>'

[Δ]<symbol name>Δ.DEFINEΔ"<character string>"

Description: This command defines a character string to a symbol.

A symbol can be redefined.

Examples:

```
        .SECTION    ram,DATA
data1:  .BLKB       1
flag    .DEFINE     "01H,data1"

        .SECTION program
        CLB         flag
```

Remarks: When defining a character string that includes a space or tab, be sure to enclose the string with single (') or double (") quotations as you write it.

Notes: The symbols defined by this directive command cannot be specified for external reference.

Section 10 Structured Description Function

as30 allows you to write the following nine types of statements.

(1) Assignment statement

The left side is substituted for by the right side.

(2) IF ELIF ELSE ENDIF statement (hereafter called the IF statement)

The IF statement changes the flow of control to two directions. The direction in which control branches is determined by a conditional expression.

(3) FOR NEXT statement (hereafter called the FOR-NEXT statement)

The FOR-NEXT statement controls repetition of operation. The statement is executed repeatedly as long as the specified conditional expression is true.

(4) FOR TO STEP NEXT statement (hereafter called the FOR- STEP statement)

The FOR-STEP statement controls the number of repetitions by specifying the initial value, an increment, and the final value.

(5) DO WHILE statement (hereafter called the DO statement)

The DO statement executes the statement repeatedly as long as the conditional expression is met (true).

(6) SWITCH CASE DEFAULT ENDS statement (hereafter called the SWITCH statement)

The SWITCH statement causes control to branch to one of the CASE blocks depending on the value of the conditional expression.

(7) BREAK statement

This statement causes the relevant FOR, DO, or SWITCH statement to stop executing, transferring control to the statement to be executed next.

(8) CONTINUE statement

This statement transfers control to a statement in the least repeated FOR or DO statement that determines repetition.

(9) FOREVER statement

This statement executes the control block repeatedly assuming that the conditional expression of the relevant FOR and DO statements are always true.

10.1 Types of Variables

In as30's structured description, the microcomputer registers and memories are referred to as variables. There are following types of variables.

(1) Register variable

This refers to the registers in the M16C Series, R8C Family microcomputers.

(2) Flag variable

This refers to the function flags of the M16C Series, R8C Family.

(3) Register bit variable

This refers to each bit position of a register variable.

(4) Memory variable

This refers to an arbitrary label or symbol.

(5) Memory bit variable

This refers to an arbitrary bit symbol.
Details on how to write each variable are explained in other sections of this manual.

(6) Reserved Variables

In as30's structured description, the register, flag, and register bit variables are processed as reserved variable names. Therefore, you cannot use a memory variable name or symbol name for the names used in these variables. For details about the register and flag functions, refer to the M16C Series, R8C Family Software Manual.

10.2 Register Variables

The table below lists the register variables. The as30 assembler does not discriminate register variable names between uppercase and lowercase letters. Consequently, 'R0L' and 'r0l' refer to the same register variable.

Table 10.1 Register Variables

Variable Name	Register Name	Variable Type Name
R0L,R0H,R1L,R1H	Data register	Byte type
R0,R1,R2,R3	Data register	Word type
A0.B,A1.B	Address register	Byte type
A0,A0.W,A1,A1.W	Address register	Word type
[A0.B],[A1.B]	Address register indirect	Byte type
[A0],[A0.W],[A1],[A1.W]	Address register indirect	Word type
[A0.A],[A1.A]	Address register indirect	Address type
[A0.L],[A1.L]	Address register indirect	Long word type
FB	Frame base register	Word type
PC	Program counter	Address type
INTBH,INTBL	Interrupt table register	Word type
INTB	Interrupt table register	Address type
SP,ISP	Stack pointer	Word type
SB	Static base register	Word type
FLG	Flag register	Word type
R2R0,R3R1	32-bit data register	Long word type
A1A0	32-bit address register	Long word type
[A1A0.B]	32-bit address register indirect	Byte type
[A1A0],[A1A0.W]	32-bit address register indirect	Word type
IPL	Processor interrupt priority level	-----

Notes SP refers to the stack pointer (user stack pointer or interrupt stack pointer) indicated by the U flag. For details about the stack pointer and U flag functions, refer to the M16C Series, R8C Family Software Manual.

10.3 Stack Variables

The table below lists the stack variables. The as30 assembler does not discriminate variable names between uppercase and lowercase letters. Consequently, 'STK' and 'stk' refer to the same variable.

Stack variables can be written for saving or restoring to or from the stack area.

Table 10.2 Stack Variables

Stack Variable Name	Content
[STK]	Memory indicated by stack pointer.

Note The stack area is indicated by the interrupt stack pointer when the U flag = 0 or the user stack pointer when the U flag = 1.

10.4 Flag Variables

The table below lists the flag variables. The as30 assembler does not discriminate flag variable names between uppercase and lowercase letters. Consequently, 'C' and 'c' refer to the same flag variable. For details about the functions of flag variables, refer to the M16C Series, R8C Family Software Manual.

Table 10.3 Flag Variables

Flag Variable Name	Flag Name
C	Carry flag
D	Debug flag
Z	Zero flag
S	Sign flag
B	Register bank specifying flag
O	Overflow flag
I	Interrupt enable flag
U	Stack pointer specifying flag

10.5 Register Bit Variables

The table below lists the register bit variables. The as30 assembler does not discriminate register bit variable names between uppercase and lowercase letters. Consequently, 'BITR0_1' and 'bitr0_1' refer to the same register bit variable.

Table 10.4 Register Bit Variables

Register Bit	Variable Name Content
BITR0_n	Bit n of data register R0 (n = 0 to 15)
BITR1_n	Bit n of data register R1 (n = 0 to 15)
BITR2_n	Bit n of data register R2 (n = 0 to 15)
BITR3_n	Bit n of data register R3 (n = 0 to 15)
BITA0_n	Bit n of data register A0 (n = 0 to 15)
BITA1_n	Bit n of data register A1 (n = 0 to 15)

Register bit variable description example)

```
Assignment statement
    BITR0_0 = 0
    BITR1_1 = 0
    BITR2_2 = 0
    BITR3_3 = 0
conditional expression
    if   BITR0_1    ;Test bit 1 of register R0
         ;
    else
         ;
    endif
```

10.6 Memory Variables

In as30's structured description, labels and symbols are processed as memory variables. The as30 assembler discriminates memory variable names between uppercase and lowercase letters.

10.6.1 Types of Memory Variables

The label and symbol names defined by the directive commands listed in the table below can be used in structured description statements as memory variables. The variable has its 'variable type' defined by the directive command.

The assembler generates object code according to the variable type.

Table 10.5 Types of Memory Variables

Assembler Directive Command	Variable Type
.BTEQU, .BTGLB	Bit type
.BLKB, .BYTE	Byte type
.BLKW, .WORD	Word type
.BLKA, .ADDR	Address
.BLKL, .LWORD	Long word type
.GLB	For externally referenced labels and symbols, write the size every line or use a command option to determine the size.

Function of command option -M

If the type of variable is not indicated when as30's command option -M is specified, the assembler assumes the byte type as it generates object code.

If this command option is not specified, the assembler assumes the word type as it generates object code.

10.6.2 Memory Variable Addressing Modes

The table below lists the address modes that can be specified in memory variables.

The addressing mode specifier (:8, :16, or :20) can be omitted.

Table 10.6 Memory Variable Addressing Modes

Addressing Mode	Addressing Mode Description Format
Absolute	[label:16], [label:20]
Address register relative	[label:8[A0]], [label:16[A0]], [label:20[A0]] [label:8[A1]], [label:16[A1]]
SB relative	[label:8[SB]], [label:16[SB]]
FB relative	[label:8[FB]]

10.6.3 Rules for Writing Memory Variables

- When writing a memory variable name in structured description statement, always be sure to enclose it with brackets [ ] or { } as you write it.
- A space or tab can be entered between the memory variable name and brackets.
- When specifying an addressing mode, always be sure to enclose it with brackets [ ] or { } along with the variable name as you write it.

Description example 1:

```
.GLB            work
.SECTION        memory,DATA
mem:            .BLKB 1
.SECTION        program,CODE
[mem] = 0
[work].B = 0
.END
```

Description example 2:

```
IF [ label[SB] ]
        ;
ELSE
        ;
ENDIF
```

10.6.4 Size Specifier

The size specifier can be set for memory variables and address register indirect addressing [A0] or [A1]. For memory variables where a size specifier is written, the assembler temporarily generates code in the specified size irrespective of the type of variable that is determined when defining a memory variable.

The table below lists the size specifiers that can be written in memory variables.

Table 10.7 Size Specifier

Size Specifier	Variable Type
.B	Byte type
.W	Word type
.A	Address type
.L	Long word type

Note The type of memory variable on a line where a size specifier is set has priority over the type determined by a directive command.

10.6.5 Rules for Writing Size Specifier

- Write a size specifier immediately after the memory variable that is enclosed with brackets.
- A space or tab can be entered between the size specifier and brackets.

Description example)

```
        .SECTION  ram,DATA
lab_b:  .BLKB     1
lab_w:  .BLKW     1
                ;
        .SECTION  rom,CODE
                ;
        [lab_b]     =  R0L    ; MOV.B   R0L,lab_b
        [lab_b].W   =  R0     ; MOV.W   R0,lab_b
        [lab_w]     =  R0     ; MOV.W   R0,lab_w
        [lab_w].B   =  R0L    ; MOV.B   R0L,lab_w
```

10.7 Memory Bit Variables

The bit symbol names defined by the directive commands listed below can be used in structured description statements as a memory bit variable.

Table 10.8 Memory Bit Variables

Assembler Directive Command	Variable Type
.BTEQU, .BTGLB	Bit type

10.7.1 Memory Bit Variable Addressing Modes

The table below lists the address modes that can be specified in memory bit variables.

- The addressing mode specifier (:8, :11, or :16) can be omitted.
- In the above table, 'bitnum' denotes a bit number and 'addr' denotes a memory address.

Table 10.9 Memory Bit Variable Addressing Modes

Addressing Mode	Addressing Mode Description Format
Absolute	[bitsym:16], [bitnum.addr:16]
SB relative	[bitsym:8[SB]], [bitnum,addr:8[SB]] [bitsym:11[SB]], [bitnum,addr:11[SB]] [bitsym:16[SB]], [bitnum,addr:16[SB]]
FB relative	[bitsym:8[FB]], [bitnum,addr:8[FB]]

Note Address register indirect and relative addressing cannot be written.

10.7.2 Rules for Writing Memory Bit Variable

- When writing a memory bit variable name in structured description statement, always be sure to enclose it with brackets [ ] or { } as you write it.
- A space or tab can be entered between the memory bit variable name and brackets.
- When specifying an addressing mode, always be sure to enclose it with brackets [ ] or { } along with the variable name as you write it.

Description example 1: For internally defined memory bit variable

```
BITSYM   .BTEQU     1,10h
if   [ BITSYM ]
        ;
else
        ;
endif
```

Description example 2: For externally referenced memory bit variable

```
.BTGLB    BITSYM
if   [ BITSYM ]
        ;
else
        ;
endif
```

10.8 Structured Operators

The following sections explain the operators that can be written in structured description statements.

(1) Unary Operators

The table below lists the unary operators that can be written in structured description statements.

Table 10.10 Unary Operators

Operator	Content
+	Represents a positive number.
-	Represents a negative number.
~	Negates every bit. (NOT)
++	Increments a single term.
--	Decrements a single term.

(2) Binary Operators

The table below lists the binary operators that can be written in structured description statements.

Table 10.11 Binary Operators

Operator	Content
+, +.C, +.D, +.CD	Adds two terms.
-, -.C, -.D, -.CD	Subtracts two terms.
*, *.S	Multiplies two terms.
/, /.S	Divides two terms.
%, %.S, %.SE	Divides two terms with residue.
&	ANDs every bit. (AND)
|	ORs every bit. (OR)
^	Exclusive ORs every bit. (EOR)
>>.C	Bit rotates the left-side value to the right by the right-side value with a carry.
<<.C	Bit rotates the left-side value to the left by the right-side value with a carry.
<>.R	Bit rotates the left-side value by the right-side value without a carry. Rotated left if the right-side value is positive; rotated right if the right-side value is negative.
<>.A	Arithmetically shifts the left-side value for a number of bits indicated by the right-side value. Shifted left if the right-side value is positive; shifted right if the right-side value is negative.
<>.L	Logically shifts the left-side value for a number of bits indicated by the right-side value. Shifted left if the right-side value is positive; shifted right if the right-side value is negative.
&&	Logically ANDs.
||	Logically ORs.

(3) Relational Operators

The table below lists the relational operators that can be written in structured description statements.

Table 10.12 Relational Operators

Operator	Content
<, <.S	Holds true when the left side is smaller than the right side.
>, >.S	Holds true when the left side is larger than the right side.
==	Holds true when the left and right sides are equal.
!=	Holds true when the left and right sides are not equal.
<=, <=.S	Holds true when the left side is smaller than or equal to the right side.
>=, >=.S	Holds true when the left side is larger than or equal to the right side.

(4) Operator Attributes

The table below lists the operator attributes specified for addition and subtraction of binary operators and in part of relational operators.

Table 10.13 Operator Attributes

Operator	Content
.C	Performs calculation with a carry or borrow.
.D	Performs decimal calculation.
.CD	Performs decimal calculation with a carry or borrow.
.S (except residue)	Performs calculation with a sign.
.S (residue)	The sign of the calculation result is made the same as that of the dividend.
.SE	The sign of the calculation result is made the same as that of the divisor.

Note No space or tab can be entered between the operator and attribute.

10.9 Expressions

There are following types of expressions.

(1) Monomial expression

An expression consisting of a single term and an expression consisting of a combination of a single term and unary operator.

(2) Binomial expression

An expression consisting of two terms and an operator.

(3) Compound expression

An expression consisting of a combination of a monomial or binomial expression and a logical operator.

10.9.1 Terms in expression

The following can be written in terms of an expression:

(1) Variable

This includes a register, flag variable, register bit variable, memory variable, and a memory bit variable.

(2) Constant

For multiplication and residue calculations, the constants shown below can be operated on.

Note Except for binary divide and residue calculations, you cannot write an expression using variables of different types.

10.9.2 Compound expression

The following shows rules for writing a compound expression.

- Up to two logical operators can be written in one expression.
- Operation on a compound expression is performed sequentially from left to right.
- A structured description command and a compound expression must be written in one line not exceeding 255 characters.
- No compound expression can be written in two or more lines.

Example of compound expression.

```
IF    [ work1 ] || [ work2 ] && [ work3 ]
            ;
ENDIF
```

10.9.3 Example of expression

The following shows examples for each type of expression. In these examples, 'mem' and 'work' denote memory variable names.

Monomial expression

```
[mem]
- [mem]
++ [mem]
```

Binomial expression

```
[mem] + 1
- [mem] + 1
```

Compound expression

```
[mem] || [work]
-- [mem] && [work]
```

10.10 Structure of Structured Description Statement

A structured description statement consists of a structured description command and a conditional expression that is written in the operand of the command. Not all structured description commands are accompanied by a conditional expression.

10.10.1 Conditional Expression

Function of conditional expression

- A conditional expression indicates a condition to be given to a structured command statement.
- Depending on whether the operation result of a conditional expression is true or false, the assembler generates object code that causes control to branch to different control blocks.

Rules for writing conditional expression

- A conditional expression can be written in the operand of a structured description command IF, ELIF, FOR(FOR-NEXT), or WHILE.
- Expressions can be written in the operand of a conditional expression.
- Always be sure to enter a space or tab between a conditional expression and a structured description command.
- When writing a structured description command and an expression, make sure that they are written in one line.
- No conditional expression can be written in two or more lines.

Description format

- Expression
- Expression Relational operator Expression
- Bit variable
- Bit variable Relational operator 1
- Bit variable Relational operator 0

Description example

The following shows a description example of a conditional expression. In this example, 'mem' and 'work' denote memory variable names; 'bit' denotes a memory bit variable name.

```
IF      [mem]
        ;
ENDIF

FOR   --[mem]
        ;
NEXT
```

```
IF      [mem] >= 0
        ;
ENDIF

FOR  [work] - [mem] <= 0
        ;
NEXT

IF      [bit]
        ;
ENDIF

IF      [bit] == 1
        ;
ENDIF

IF      [bit] != 0
        ;
ENDIF
```

10.10.2 Nesting of Structured Description Statements

Structured description statements can be nested in up to a total of 65,535 levels. However, no intertwined nesting of statements like the example shown below are accepted.

Furthermore, no intertwined nesting of statements including macro directive commands or assembler directive commands .IF, .ELIF, .ELSE, or .ENDIF are accepted.

Example of incorrect (intertwined) nesting

```
FOR  R0 = 1 TO 10 STEP 1
            ;
        IF  R1 == 3      ;The if statement begins in a for statement.
            ;
NEXT
        ENDIF            ; The if statement ends outside the for statement.
```

10.11 List of Structured Description Commands

The following pages show rules for writing structured description commands.

Condition branch

IF Statement

Format:(IF-ENDIF)

```
IFΔ<conditional expression>
        Control block
ENDIF
```

Description: The basic structure of an IF statement consists of structured description commands IF and ENDIF and a control block enclosed with these commands.

Control branches to ENDIF if the condition of IF is false.

Remarks: Expressions described in "10.10.1 Conditional Expression" can be used the conditional expression.

Format:(ELSE)

```
IFΔ<conditional expression>
        Control block
ELSE
        Control block
ENDIF
```

Description: Structured description command ELSE can be written in the IF statement.

If the conditional expression of IF is false, control branches to the control block that follows ELSE.

If there are two or more control blocks, branching to ENDIF occurs at the end of each control block.

Remarks: Only one instance of ELSE can be written between IF and ENDIF.

Format:(ELIF)

```
IFΔ<conditional expression>
        Control block
ELIFΔ<conditional expression>
        Control block
ENDIF
```

Description: Structured description command ELIF can be written in the IF statement.

If the conditional expression of IF is false, the assembler checks the conditional expression of ELIF to see if it is true or false.

If the conditional expression of ELIF is true, control branches to the beginning of the immediately following control block.

If the conditional expression of ELIF is false, control branches to the immediately following structured description command (ELIF, ELSE, or ENDIF).

If there are two or more control blocks, branching to ENDIF occurs at the end of each control block.

Remarks: Expressions described in "10.10.1 Conditional Expression" can be used the conditional expression.

More than one instance of ELIF can be written between IF and ELSE or between IF and ENDIF.

Example:

```
IF      [sym1] == 10
        ;
ELIF [sym2] != 10
        ;
ELSE
        ;
ENDIF
```

Expansion example:

```
        CMP.B       #10,sym1
        JNE         ..IF0002
        ;
        JMP         ..IF0003
..IF0002:
        CMP.B       #10,sym2
        JEQ         ..IF0004
        ;
        JMP         ..IF0003
..IF0004:
        ;
..IF0003:
```

Repeat (condition)

FOR-NEXT Statement

Format:

```
FORΔ<conditional expression>
        Control block
ENDIF
```

Description: The basic structure of a FOR statement consists of structured description commands FOR and NEXT and a control block enclosed with these commands.

If the conditional expression is true, control branches to the immediately following control block.

If the conditional expression is false, control branches to a line that immediately follows the structured description command NEXT.

A BREAK statement can be written in the control block. This BREAK statement forcibly terminates repetition control.

A CONTINUE statement can be written in the control block. This CONTINUE statement causes control to branch to the NEXT statement.

A FOREVER statement can be written in the conditional expression. This FOREVER statement continues executing the control block repeatedly.

Example:

```
FOR R0 <.S 10
      ;
NEXT
```

Expansion example:

```
; Repeated as long as R0 is smaller than 10
..fr0000:
        CMP.W     #10,R0
        JGE       ..fr0002
        ;
        JMP       ..fr0000
..fr0002
```

Remarks: Expressions described in "10.10.1 Conditional Expression" can be used the conditional expression.

Repeat (loop counter)

FOR-STEP Statement

Format:

```
FORΔ<Loop counter>=<Initial value>TOΔ<final value>[Δ STEPΔ<increment>]
        Control block
ENDIF
```

Description: The basic structure of a FOR statement consists of structured description commands FOR and NEXT and a control block enclosed with these commands.

The loop counter value specified in the operand of structured description command FOR is updated for a specified amount of increment. When the value becomes equal to the final value, the control block is executed.

If the loop counter value equals the final value, control branches to the line immediately following structured description command NEXT.

If the specified increment is a negative value, the loop counter is counted down.

If the increment is omitted, the assembler assumes '+1' as it generates object code.

A BREAK statement can be written in the control block. This BREAK statement forcibly terminates repetition control.

A CONTINUE statement can be written in the control block. This CONTINUE statement causes control to branch to the NEXT statement.

Example:

```
FOR [lab].W = 0 TO 10 STEP 1    ;lab is initialized to 0 which is repeated up to 10
     ;
NEXT
```

Expansion example:

```
     MOV.W    #0,lab
..fr0000:
     CMP.W    #10,lab
     JEQ      ..fr0002
     ;
..fr0001:
     ADD.W    #1,lab
     JMP      ..fr0000
..fr0002:
```

Remarks: A register variable and memory variable can be written in the loop counter.

Variables or constant values can be used in the initial and final values.

A constant value can be used in the increment.

A local symbol name can be written as a constant value.

Notes: The control block is always repeated until the loop counter value becomes equal to the final value.

If the register variable or memory variable used in the loop counter has its content modified in the control block, the FOR statement will not be executed correctly.

Multi condition

SWITCH Statement

Format: (Basic configuration)

```
SWITCHΔ<expression>
        CASEΔ<data>
              Control block
        CASEΔ<data>
              Control block
ENDS
```

Description: The basic structure of a SWITCH statement consists of structured description commands SWITCH, ENDS, and CASE and a control block enclosed with CASE statement.

Control branches to a control block immediately following the CASE command that holds data that matches the content of the expression written in the operand of the SWITCH statement.

Evaluation is made on all CASE command data.

Remarks: Unary operators and Binary operators expressions described in '10.8 Expressions' can be written in the operand expression of SWITCH.

Be sure to write more than one instance of CASE statement. If no CASE is found between SWITCH and ENDS, the assembler outputs a warning.

A constant can be written in the operand data of CASE.

No value can be written in the operand data of CASE unless the value is fixed when assembled.

No values can be written in the operand data of CASEs that are the same in one SWITCH statement.

Format: (BREAK)

```
SWITCHΔ<expression>
CASEΔ<data>
        Control block
        BREAK
CASEΔ<data>
        Control block
ENDS
```

Description: The BREAK statement causes control to branch to ENDS unconditionally.

Remarks: The BREAK command must be written at the end of a control block.

If this command is written in the middle of a control block, the assembler outputs a warning. In this case, although code for lines between the BREAK command and the next structured description command is generated, no code is generated for branching to that section.

Format: (DEFAULT)

```
SWITCHΔ<expression>
CASEΔ<data>
        Control block
CASEΔ<data>
        Control block
DEFAULT
        Control block
ENDS
```

Description: If no matching data is found in the expression, control branches to the control block that immediately follows DEFAULT.

A warning is output for CASE that is written between structured description command DEFAULT and ENDS. In this case, although object code for the control block immediately following this instance of CASE is generated, no code is generated for branching to that block.

Remarks: A structured description command 'DEFAULT' and a control block can be written at a position immediately preceding ENDS of a SWITCH statement.

Only once instance of structured description command DEFAULT can be written in one SWITCH statement.

Example:

```
SWITCH [ work ]
     CASE 1
          ;
          BREAK
     CASE 2
          ;
     DEFAULT
          ;
ENDS
```

Expansion example:

```
     CMP.B   #1,work        ; Generated for CASE.
     JNE     ..sw0004       ; Generated for CASE.
     ;
     JMP     ..sw0000       ; Generated for BREAK.
..sw0004:                   ; Generated for CASE.
     CMP.B   #2,work        ; Generated for CASE.
     JNE     ..sw0006       ; Generated for CASE.
     ;
..sw0006:                   ; Generated for DEFAULT.
     ;
..sw0000:                   ; Generated for ENDS.
```

Repeat(condition)

DO-WHILE Statement

Format:

```
DO
        Control block
WHILEΔ<conditional expression>
```

Description: The basic structure of a DO statement consists of structured description commands 'DO' and 'WHILE' and a control block enclosed with these commands.

After executing the control block, the assembler checks the conditional expression written in the operand of WHILE to see if it is true or false.

If the conditional expression is true, control branches to DO.

If the conditional expression is false, control branches to the next line.

A BREAK statement can be written in the control block. This BREAK statement causes control to branch to the line next to WHILE.

A CONTINUE statement can be written in the control block. This CONTINUE statement causes control to branch to the WHILE statement.

A FOREVER statement can be written in the conditional expression. This statement causes control to branch to the DO statement unconditionally.

Example:

```
DO
        ;
WHILE [lab].b ==1
```

Expansion example:

```
..DO0000:
        ;
        CMP.B   #1,lab
        JEQ     ..DO0000
..DO0002:
```

Remarks: Expressions described in "10.10.1 Conditional Expression" can be used the conditional expression.

Unconditional branch

BREAK

Format: BREAK

Description: This statement generates an unconditional branch instruction.

Example:

```
FOR [lab]=1 TO 10 STEP 1
                ;
        BREAK
                ;
NEXT
```

Expansion example:

```
        MOV.W     #1,lab          ; Generated for FOR.
..fr0000:                         ; Generated for FOR.
        CMP.W     #10,lab         ; Generated for FOR.
        JEQ       ..fr0002        ; Generated for FOR.
        ;
        JMP       ..fr0002        ; Generated for BREAK.
        ;
..fr0001:                         ; Generated for STEP.
        ADD.W     #1,lab          ; Generated for STEP.
        JMP       ..fr0000        ; Generated for STEP.
..fr0002:                         ; Generated for NEXT.
```

Remarks: A BREAK statement can be written in the control block of FOR, DO, and SWITCH.

A BREAK statement can be written in the control block of an IF statement providing that it exists in the control block of FOR, DO, or SWITCH statement.

No BREAK statement can be written in the control block of an ordinary IF statement.

Unconditional branch (condition sentence)

CONTINUE

Format: CONTINUE

Description: This statement generates an unconditional branch instruction.

Example:

```
FOR [lab]=1 TO 10 STEP 1
                ;
        CONTINUE
                ;
NEXT
```

Expansion example:

```
        MOV.W     #1,lab          ; Generated for FOR.
..fr0000:                         ; Generated for FOR.
        CMP.W     #10,lab         ; Generated for FOR.
        JEQ       ..fr0002        ; Generated for FOR.
        ;
        JMP       ..fr0001        ; Generated for CONTINUE.
;
..fr0001:                         ; Generated for STEP.
        ADD.W     #1,lab          ; Generated for STEP.
        JMP       ..fr0000        ; Generated for STEP.
..fr0002:                         ; Generated for NEXT.
```

Remarks: A CONTINUE statement can be written in the control block of FOR and DO statements.

A CONTINUE statement can be written in the control block of an IF or SWITCH statement providing that it exists in the control block of FOR or DO.

No CONTINUE statement can be written in the control block of an ordinary IF or SWITCH statement.

Unconditional branch (condition sentence)

FOREVER

Format1:

```
FORΔFOREVER
        Control block
NEXT
```

Format2:

```
DO
        Control block
WHILEΔFOREVER
```

Description: This command continues executing the control block repeatedly.

A BREAK statement can be written in the control block. This BREAK statement forcibly terminates repetition control.

A CONTINUE statement can be written in the control block. This CONTINUE statement causes control to branch to a statement that determines whether or not to repeat.

Example:

```
FOR   FOREVER
        ;
NEXT
```

Expansion example:

```
..fr0000:
        ;
        JMP     ..fr0000
..fr0002:
```

Assignment Statement

Format: <left side>=<right side>

Description: An assignment statement substitutes the calculation result of the expression on the right side of the statement for a variable on the left side.

There are following types of assignment statements.

Table 10.14 types of assignment statements

Operations	content
=	Substitutes an unsigned value for the left side.
=.S	Substitutes a sign-extended value on the right side for the left side.
=.Z	Substitutes a zero-extended value on the right side for the left side.
=.EL	Generates a LDE command.
=.ES	Generates a STE command.

Remarks: No expressions that contain unary or binary operators can be written on the right side of assignment statement '=.S', '=.Z', '=.EL', or '=.ES'.

Variables listed below can be written on the left and right sides of assignment statements '=.S' and '=.Z':

Memory variables (except for [SP] relative)

Data register and address register indirect among register variables

The variables that can be written on the left and right sides of assignment statement '=.EL' are those whose contents can be written in the operands 'dest' and 'src' of mnemonic LDE.

The variables that can be written on the left and right sides of assignment statement '=.ES' are those whose contents can be written in the operands 'dest' and 'src' of mnemonic STE.

A warning is output if an entirely same variable is written on the left and right sides of an assignment statement.

If a different type of variable is substituted for, no expressions can be written on the right side of the assignment statement that contains unary or binary operators.

Note: For details about mnemonics, refer to the 'M16C Series, R8C Family Software Manual'.

Combination of variable types that can be written in assignment statement (=)

Table 10.15 variable types

Left side(Type)	**Right side(Type)**			
	Byte	**Word**	**Address**	**Long word**
Byte	O	×	×	×
Word	×	O	×	×
Address	×	×	O	×
Long word	×	×	×	O

Combination of variable types that can be written in sign-extended assignment statement (=.S)

Table 10.16 sign-extended assignment statement

Left side(Type)	**Right side(Type)**			
	Byte	**Word**	**Address**	**Long word**
Byte	×	×	×	×
Word	O	×	×	×
Address	×	×	×	×
Long word	×	O	×	×

Notes: If for a 'word type =.S byte type' assignment expression, R2 or R3 is specified for the left side of the expression, the assembler uses the R0 register.
If for a 'long word type =.S word type' assignment expression, memory variable or R3R1 is specified for the left side of the expression, the assembler uses the R2R0 register pair.

Combination of variable types that can be written in zero-extended assignment statement (=.Z)

Table 10.17 zero-extended assignment statement

Left side(Type)	**Right side(Type)**			
	Byte	**Word**	**Address**	**Long word**
Byte	×	×	×	×
Word	O	×	×	×
Address	O	O	×	×
Long word	O	O	O	×

Note: If for a 'word type =.Z byte type' assignment expression, R2, R3 is specified for the right side of the expression, the assembler uses the R0 register.

Combination of variable types that can be written in special assignment statements (=.EL, =.ES)

Table 10.18 special assignment statement

Left side(Type)	**Right side(Type)**			
	Byte	**Word**	**Address**	**Long word**
Byte	O	×	×	×
Word	×	O	×	×
Address	×	×	×	×
Long word	×	×	×	×

Description example of assignment statement and its expansion example

Table 10.19 assignment statement

Example of source description	**Expansion example**	
R1 = R0	MOV.W	R0,R1
R0 = R0 + 2	ADD.W	#2,R0
R0 =.S R0L	EXTS.B	R0L
R0 =.Z R0L	MOV.B	#0,R0H
R0L =.EL [lab].B	LDE.B	lab,R0L
[lab].W =.ES R0	STE.W	R0,lab
R0 =.S R0H	MOV.B	R0H,R0L
	EXTS.B	R0L
[lab_w].W =.S R0L	MOV.B	R0L,lab_w
	EXTS.B	lab_w
R2R0 =.S R0	EXTS.W R0	
R2R0 =.S R1	MOV.W	R1,R0
	EXTS.W R0	
[lab_l].L =.S R0	EXTS.W R0	
	MOV.W	R0,lab_l
	MOV.W	R2,lab_l+2
R0 =.Z R0H	MOV.B	R0H.R0L
	MOV.B	#0,R0H
[lab_w].W =.Z R0L	MOV.B	R0H,lab_w
	MOV.B	#0,lab_w+1
[lab_a].A =.Z R0	MOV.W	R0L,lab_a
	MOV.B	#0,lab_a+2

10.12 Structure of Structured Description Commands

This section shows structured description statements that can be written in as30 programming. When writing structured description, please follow the syntax shown below.

Definition of Terms

The following explains the description terms used in this section. The variable name or operator indicated by each term can be written at the position where the term is written.

(1) Register variable

Table 10.20 Register variable

Term	Contents
regb	R0L, R0H, R1L, R1H, A0.B, A1.B, [A0.B], [A1.B]
regw	R0, R1, R2, R3, A0, A1, [A0], [A1]
regc	FB, SB, SP, ISP, FLG, INTBH, INTBL
reglw	R2R0, R3R1
regad	A1A0

Notes: SP refers to the stack pointer (user stack pointer or interrupt stack pointer) indicated by the U flag. For details about the stack pointer and U flag functions, refer to the 'M16C Series, R8C Family Software Manual'.

(2) Memory variable

Table 10.21 Memory variable

Term	Contents
memb	Byte type memory variable (except for description of 'SP')
memw	Word type memory variable (except for description of 'SP')
mema	Address type memory variable
meml	Long word type memory variable
regmembit	Register bit variable, memory bit variable
flgbit	Flag variable

(3) Operators

Table 10.22 Operators

Term	Contents
Unary operators	~, -, ++, --
Binary operators 1	+, +.C, -, -.C
Binary operators 2	+.C, +.CD, -.C, -.CD
Binary operators 3	*, *.S
Binary operators 4	/, /.S, %, %.S, %.SE
Binary operators 5	&, |, ^, ?
Binary operators 6	>>.C, <<.C
Binary operators 7	<>.R
Binary operators 8	<>.A, <>.L
Relational operators	==, !=, >, >.S, <, <.S, =>, =>.S, <=, <=.S
Coincidence comparing operators	==, !=
Logical operators	&&, ||
Constants	Numeric value or expression value that is fixed when assembled

10.13 Syntax of Statements

The following shows the syntax of statements.

Uo = Unary operator
Bo = Binary Operator
Ro = Relational operator
Co = Coincidence comparing operator
Lo = Logical operator

(1) Simple assignment statements and assignment statements containing unary operators

Note: Only the data register variables can be written for 'regb' and 'regw' in '=.S' and '=.Z'.

- Left side is Memory variable

```
memb    =     <constant>
memb    =     <Uo> memb
memb    =     <Uo> regb
memw    =     <constant>
memw    =.S   <Uo> memw
memw    =.S   <Uo> regw
memw    =.S   memb
memw    =.S   regb
memw    =.Z   memb
memw    =.Z   reg
mema    =     <constant>
mema    =     mema
mema    =.Z   memb
mema    =.Z   memw
mema    =.Z   regb
mema    =.Z   regw
```

memlw	=	<constant>
memlw	=	meml
memlw	=	R2R0
memlw	=	R3R1
memlw	=	A1A0
memlw	=.S	memw
memlw	=.S	regw
memlw	=.Z	memb
memlw	=.Z	memw
memlw	=.Z	mema
memlw	=.Z	regb
memlw	=.Z	regw
memb	=.ES	memb,regb
memw	=.ES	memw,regw
memb	=	[STK].B
memw	=	[STK].W
dsp:8[SP]	=	memb,regb
dsp:8[SP]	=	memw,regw

- Left side is Register

regb	=	<constant>
regb	=	<Uo> memb
regb	=	<Uo> regb
regw	=	<constant>
regw	=	<Uo> memw
regw	=	<Uo> regw
regw	=.S	memb
regw	=.S	regb
regw	=.Z	memb
regw	=.Z	regb
regl	=	<constant>
regl	=	meml
regl	=	R2R0
regl	=	R3R1
regl	=	A1A0
regl	=.S	memw
regl	=.S	regw
regl	=.Z	memb
regl	=.Z	memw
regl	=.Z	mema
regl	=.Z	regb
regl	=.Z	regw
regc	=	<constant>
regc	=	memw
regc	=	regw
regb(Except for A0.B and A1.B)= [STK].B		
regw	=	[STK].W
regc	=	[STK].W
R0,R1,R2,R3, A0,A1,SB,FB=[STK].W(Multiple register can be written in left side)		
INTB	=	<constant>
IPL	=	<constant>

- Left side is Register or Memory variable

memb, regb	=.EL	memb
memw, regw	=.EL	memw
memw, regw	=	regc
memb,regb	=	dsp:8[SP]
memw,regw	=	dsp:8[SP]
mema, [A0.A], [A1.A], R2R0, R3R1, A1A0	=	regpc

- Left side is Stack Variables

[STK].B	=	<constant>
[STK].B	=	memb
[STK].B	=	regb (Except for A0.B and A1.B)
[STK].W	=	<constant>
[STK].W	=	memw
[STK].W	=	regw
[STK].W	=	regc
[STK].W	=	R0,R1,R2,R3,A0,A1,SB,FB (Multiple register can be written)
[STK].A	=	mema

- Left side is bit variable

regmembit	=	1, 0, ~regmembit(Bit name is same as left side)
flgbit	=	1, 0

- Assignment statements containing unary operators

memb/regb	=	Uo memb/regb
memw/regw	=	Uo memw/regw

- Assignment statements containing binary operators 1

memb/regb	=	[Uo] memb/regb Bo 1 constant/memb/regb
memw/regw	=	[Uo] memw/regw Bo 1 constant/memw/regw

- Assignment statements containing binary operators 2

memb/regb	=	[Uo] memb/regb Bo 2 constant/memb/regb
memw/regw	=	[Uo] memw/regw Bo 2 constant/memw/regw

- Assignment statements containing binary operators 3

memw/regw	=	[Uo] memb/regb Bo 3 constant/memb/regb
meml/regl	=	[Uo] memw/regw Bo 3 constant/memw/regw

- Assignment statements containing binary operators 4

memb/regb	=	[Uo] memb/regb Bo 4 constant/memb/regb
memw/regw	=	meml/reglw/regad Bo 4 constant/memw/regw

- Assignment statements containing binary operators 5

memb/regb	=	[Uo] memb/regb Bo 5 constant/memb/regb
memw/regw	=	[Uo] memw/regw Bo 5 constant/memw/regw

- Assignment statements containing binary operators 6

memb/regb	=	[Uo] memb/regb Bo 6 constant
memw/regw	=	[Uo] memw/regw Bo 6 constant

- Assignment statements containing binary operators 7

memb/regb	=	[Uo] memb/regb Bo 7 constant/R1H
memw/regw	=	[Uo] memw/regw Bo 7 constant/R1H

- Assignment statements containing binary operators 8

memb/regb	=	[Uo] memb/regb Bo 8 constant/R1H
memw/regw	=	[Uo] memw/regw Bo 8 constant/R1H
meml/reglw/regad	=	meml/reglw/regad Bo 8 constant/R1H

(2) Syntax of expression 1

[Uo] memb/regb		
[Uo] memw/regw		
Expression 2		
Expression 2	Ro	Immediate/memb/regb
Expression 2	Ro	Immediate/memw/regw
Expression 2	Lo	Expression 2
Expression 3	Lo	Expression 3
Expression 3		
regmembit/flgbit		

(3) Syntax of expression 2

Among syntaxes indicated on the right side of the assignment expression, all syntaxes except for the following contents can be written.

- Registers and stacks listed below

  FB, SB, SP, ISP, FLG, INTBH, INTBL, INTB, IPL and [STK]

- Expressions where multiplication results in 32 bits

- Inverted expressions of register bit and memory bit variables

  ~regmembit

(4) Syntax of expression 3

Binomial expression .b	Ro	Constant/memb/regb
Binomial expression .w	Ro	Constant/memw/regw
regmembit/flgbit	=	coincidence comparing operator 1/0

(5) Syntax of Conditional Expression

IF statement
```
IF          Expression 1
```

FOR-NEXT statement
```
FOR         Expression 1
```

FOR-STEP statement
```
FOR         variable= [Uo]variable/constant   TO   variable/constant   STEP
```

WHILE statement
```
WHILE       Expression 1
```

SWITCH statement
```
SWITCH      Expression
```

Section 11 Error Messages for the Assembler

11.1 Error Format and Error Levels

This section gives a list of error messages and explains details of errors in the following format.

Error number (Error level) **Error message**
Error details

There are four different error levels, corresponding to different degrees of seriousness.

Error Number	Error Level	Error Type	Description
A1000 – A1999	(W)	Warning	Processing is continued.
A2000 – A2999	(E)	Error	Processing is interrupted.
A3000 – A3999	(F)	Fatal	Processing is interrupted.
A4000 – A4999	(-)	Internal	Processing is interrupted.

11.2 Return Values for Errors

When terminating execution, each as30 program returns a numeric value to the OS indicating its status at termination. The table below lists the values that are returned when an error is encountered.

Return value	Content
0	Program terminated normaly.
1	Program was forcibly terminated by input of control C.
2	Error relating to the OS's file system or memory system occured.
3	Error attributable to the file being processed occured.
4	Error in input form the command line occured.

11.3 List of Messages

A1001 (W) Non support command option 'xxx' is used

An unsupported command option is set.
Reenter the command option.

A1101 (W) Too many actual macro parameters

There are too many actual macro parameters.
Extra macro parameters will be ignored.

A1102 (W) Actual macro parameters are not enough

The number of actual macro parameters is smaller than that of formal macro parameters.
The formal macro parameters that do not have corresponding actual macro parameters are ignored.

A1103 (W) String 'xxx' is too long

The character string is excessively long.
Limit the length of the character string.

A1104 (W) Symbol 'xxx' is not defined (regarded as 0)

An undefined symbol is used. It is assumed to be 0 when processed.
Define the symbol.

A1105 (W) Unnecessary ':' is found

The macro name is followed by a colon.
Delete the colon inserted after the macro name. Use a command option -I to have it ignored.

A1106 (W) Source line exceeds 8192 characters

The line-concatenated or macro argument-converted source lines contain more than 8192 characters.
Make sure the number of characters in these source lines do not exceed 8192.

A1107 (W) .END statement is in include file

The include file contains an '.END' statement.
'.END' cannot be written in include files. Delete this statement.
The software will ignore '.END' as it executes.

A1200 (W) '.ALIGN' with not 'ALIGN' specified relocatable section

Directive command '.ALIGN' is written in a section that does not have an ALIGN
Check the position where directive command '.ALIGN' is written.
Write an ALIGN specification in the section definition line of a section in which directive command '.ALIGN' is written.

A1201 (W) Destination address may be changed

The jump address can be a position that differs from an anticipated destination.
When writing an address in a branch instruction operand using a location symbol for offset, be sure to write the addressing mode, jump distance, and instruction format specifiers for all mnemonics at locations from that instruction to the jump add

A1202 (W) Floating point value is out of range

The floating-point number is out of range.
Check whether the floating-point number is written correctly. Values out of range will be ignored.

A1203 (W) Location counter exceed

The location counter exceeded xxx.
Check the operand value of '.ORG' Rewrite the source correctly.

A1204 (W) Moved between address registers as byte size

Transfers between address registers are performed in bytes.
Rewrite the mnemonic correctly.

A1205 (W) Invalid '.SBSYM' declaration, it's declared by '.FBSYM'

The symbol is already declared in '.FBSYM'. The '.SBSYM' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1206 (W) Invalid '.FBSYM' declaration, it's declared by '.SBSYM'

The symbol is already declared in '.SBSYM'. The '.FBSYM' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1207 (W) Addressing is described by the numerical value

Addressing is specified with a numeric value.
Be sure to write '#' in numeric values.

A1208 (W) The shift instruction which uses R1H is described

The shift number of times of the shift instruction is set in R1H.
Confirm whether or not it doesn't correspond to the attention item of the device.

A1209 (W) Mnemonic in 'ROMDATA' section

Found mnemonic in the section type is ROMDATA.
Specify CODE type to the section written mnemonic.

A1210 (W) Fixed data in 'CODE' section

Found directive command (.BYTE, .WORD(S), .ADDR, .LWORD) in the section type is CODE.
Specify ROMDATA type the section written any directive command (.BYTE, .WORD(S), .ADDR, .LWORD).

A1211 (W) Control register differ size

The control register is a different size than that of the M16C/80 Series and other MCU's of the M16C/60 Family.
Match the data size of the operand to the control register size of the M16C/80 Series.

A1212 (W) Calculation result is different

The calculation result is different.
Confirm a calculation result.

A1213 (W) Invalid '.FBSYM' declaration, it's declared by '.SBSYM'

The symbol is already declared in '.SBSYM'. The '.FBSYM' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1214 (W) Invalid '.SBSYM16' declaration, it's declared by '.SBSYM'

The symbol is already declared in '.SBSYM'. The '.SBSYM16' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1215 (W) Invalid '.SBSYM' declaration, it's declared by '.FBSYM'

The symbol is already declared in '.FBSYM'. The '.SBSYM' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1216 (W) Invalid '.SBSYM16' declaration, it's declared by '.FBSYM'

The symbol is already declared in '.FBSYM'. The '.SBSYM16' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1217 (W) Invalid '.SBSYM' declaration, it's declared by '.SBSYM16'

The symbol is already declared in '.SBSYM16'. The '.SBSYM' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1218 (W) Invalid '.FBSYM' declaration, it's declared by '.SBSYM16'

The symbol is already declared in '.SBSYM16'. The '.FBSYM' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1219 (W) '-JOPT' and '.OPTJ' are specified

-JOPT option and the directive command '.OPTJ' are specified.
The directive command '.OPTJ' is ignored.

A1220 (W) '.ALIGN' size is different

The size of alignment correction values is different.
Check the size of alignment correction value.

A1221 (W) Fixed point value is out of range

The Fixed-point number is out of range.
Check whether the fixed-point number is written correctly.
Values out of range will be ignored.

A1222 (W) The register used by the operation is different

The written instruction has its functionality altered due to MCU change.
Check the functionality of the instruction.

A1223 (W) Use string instructions

String instruction is used.
Confirm whether or not it doesn't correspond to the attention item of the device.

A1224 (W) Use product sum operation instruction

Sum-of-products instructions is used.
Confirm whether or not it doesn't correspond to the attention item of the device.

A1225 (W) Invalid '.SB_AUTO_SBSYM' declaration, it's declared by '.FBSYM'

The symbol is already declared in '.FBSYM'. The '.SB_AUTO_SBSYM' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1226 (W) Invalid '.FBSYM' declaration, it's declared by '.SB_AUTO_SBSYM'

The symbol is already declared in '.SB_AUTO_SBSYM'. The '.FBSYM' declaration will be ignored.
Rewrite the symbol declaration correctly.

A1227 (W) Section attribute mismatch

The section attribute is incorrect.
Make sure the section type and ALIGN setting agree.

A1228 (W) Non support directive command is used

An unsupported directive command is set.
Rewrite the declaration.

A1229 (W) Invalid '.SECTION' declaration

Sections cannot be declared. The definition will be ignored.
Rewrite the declaration.

A1230 (W) Function information is not defined

Function information, which is inspector information, has not been defined.
Define the function information as required.

A1300 (W) Statement has not effect

The statement does not have any effect as a command line.
Check the correct method for writing the command.

A1301 (W) 'CASE' not exist in 'SWITCH' statement

No CASE description is found in the SWITCH statement.
Make sure the SWITCH statement contains at least one CASE statement.

A1303(W) 'CASE' definition is after 'DEFAULT'

CASE is preceded by a DEFAULT description.
Make sure all DEFAULT commands are written after the CASE statement.

A1304 (W) Bit number is ignored

Bit numbers cannot be specified. The bit number will be ignored.
Check the written content.

A1305 (W) Too many structured label definition

There are too many labels to be generated.
Divide the file into smaller files before assembling.

A1306 (W) Unnecessary BREAK is found

Found two or over BREAK statement in a SWITCH block.
Check the source program.

A2001 (E) No input files specified

No input file is specified.
Specify an input file.

A2002 (E) Invalid option 'option' is used

An invalid command option 'option' is used.
The specified option is nonexistent. Re-input the command correctly.

A2003 (E) Option 'option' is not appropriate

Command option 'option' is written incorrectly.
Specify the command option correctly again.

A2004 (E) Source files number exeed 80

The number of source files exceeds 80.
Execute assembling separately in two or more operations.

A2005 (E) Command line is too long

The command line has too many characters.
Re-input the command.

A2006 (E) Specified an option that can't be used with '-R8C'

The option that cannot be used with -R8C at the same time is specified.
Check the option.

A2007 (E) Specified an option that can't be used with '-R8CE'

The option that cannot be used with -R8CE at the same time is specified.
Check the option.

A2101 (E) No .END statement

'.END' is not entered.
Be sure to enter '.END' in the last line of the source program.

A2102 (E) Value is out of range

The value is out of range.
Write a value that matches the register bit length.

A2103 (E) Illegal operand is used

The operand is incorrect.
Check the syntax for this operand and rewrite it correctly.

A2104 (E) Illegal directive command is used

An illegal instruction is entered.
Rewrite the instruction correctly.

A2105 (E) Invalid label definition

An invalid label is entered.
Rewrite the label definition.

A2106 (E) No ';' at the top of comment

';' is not entered at the beginning of a comment.
Enter a semicolon at the beginning of each comment.
Check whether the mnemonic or operand is written correctly.

A2107 (E) Invalid symbol definition

An invalid symbol is entered.
Rewrite the symbol definition.

A2108 (E) Include nesting over

Include is nested too many levels.
Rewrite include so that it is nested within the valid levels.

A2109 (E) Can't open include file 'filename'

The include file cannot be opened.
Check the include file name. Check the directory where the include file is stored.

A2110 (E) Can't open '.ASSERT' message file 'filename'

The '.ASSERT' output file cannot be opened.
Check the file name.

A2111 (E) Can't write '.ASSERT' message file 'filename'

Data cannot be written to the '.ASSERT' output file.
Check the permission of the file.

A2112 (E) Including the include file in itself

An attempt is made to include the include file in itself.
Check the include file name and rewrite correctly.

A2113 (E) Too many macro nesting

The macro is nested too many levels.
Make sure that the macro is nested no more than 65,535 levels .
Check the syntax for this source statement and rewrite it correctly.

A2114 (E) Too many macro local label definition

Too many macro local labels are defined.
Make sure that the number of macro local labels defined in one file are 65,535 or less.

A2115 (E) Operand number is not enough

The number of operands is insufficient.
Check the syntax for these operands and rewrite them correctly.

A2116 (E) Reserved word is used as label or symbol

Reserved word is used as a label or symbol.
Rewrite the label or symbol name correctly.

A2117 (E) ')' is missing

')' is not entered.
Write the right parenthesis ')' corresponding to the '('.

A2118 (E) '.IF' is missing for '.ELSE'

'.IF' for '.ELSE' is not found.
Check the position where '.ELSE' is written.

A2119 (E) '.IF' is missing for '.ELIF'

'.IF' for '.ELIF' is not found.
Check the position where '.ELIF' is written.

A2120 (E) '.IF' is missing for '.ENDIF'

'.IF' for '.ENDIF' is not found.
Check the position where '.ENDIF' is written.

A2121 (E) '.MACRO' is missing for '.ENDM'

'.MACRO' for '.ENDM' is not found.
Check the position where '.ENDM' is written.

A2122 (E) '.MREPEAT' is missing for '.ENDR'

'.MREPEAT' for '.ENDR' is not found.
Check the position where '.ENDR' is written.

A2123 (E) '.MACRO' or '.MREPEAT' is missing for '.EXITM'

'.MACRO' or '.MREPEAT' for '.EXITM' is not found.
Check the position where '.EXITM' is written.

A2124 (E) No macro name

No macro name is entered.
Write a macro name for each macro definition.

A2125 (E) Symbol is multiple defined

The symbol is defined twice or more. The macro name and some other name are duplicates.
Change the name.

A2126 (E) Too many formal parameter

There are too many formal parameters defined for the macro.
Make sure that the number of formal parameters defined for the macro is 80 or less.

A2127 (E) Illegal macro parameter

The macro parameter contains some incorrect description.
Check the written contents of the macro parameter.

A2128 (E) Source line is too long

The source line is excessively long.
Check the contents written in the source line and correct it as necessary.

A2129 (E) '.MACRO' is missing for '.LOCAL'

'.MACRO' for '.LOCAL' is not found.
Check the position where '.LOCAL' is written. '.LOCAL' can only be written in a macro block.

A2130 (E) Too many nesting level of condition assemble

Condition assembling is nested too many levels.
Check the syntax for this condition assemble statement and rewrite it correctly.

A2131 (E) No '.ENDM' statement

'.ENDM' is not entered.
Check the position where '.ENDM' is written. Write '.ENDM' as necessary.

A2132 (E) No '.ENDR' statement

'.ENDR' is not entered.
Check the position where '.ENDR' is written. Write '.ENDR' as necessary.

A2133 (E) Symbol is undefined

The symbol is not defined yet.
Undefined symbols cannot be used. Forward referenced symbol names cannot be entered.
Check the symbol name.

A2134 (E) No .ENDIF statement

'.ENDIF' is not entered.
Check the position where '.ENDIF' is written. Write '.ENDIF' as necessary.

A2135 (E) Division by zero

A divide by 0 operation is attempted.
Rewrite the expression correctly.

A2136 (E) Quote is missing

Quotes for a character string are not entered.
Enclose a character string with quotes as you write it.

A2137 (E) Right quote is missing

A right quote is not entered.
Enter the right quote.

A2138 (E) '{' is missing

'{' is not entered.
Write the parenthesis '{' corresponding to the '}'.

A2139 (E) The value is not constant

The value is indeterminate when assembled.
Write an expression, symbol name, or label name that will have a determinate value when assembled.

A2140 (E) Too many temporary label

There are too many temporary labels.
Replace the temporary labels with label names.

A2141 (E) Temporary label is undefined

There are too many temporary labels.
Replace the temporary labels with label names.

A2142 (E) Syntax error in expression

The expression is written incorrectly.
Check the syntax for this expression and rewrite it correctly.

A2143 (E) Symbol is expected

Symbols are insufficient.
Check the number of symbols.

A2144 (E) Illegal macro statements

Directive command '.IF' and nesting are crossing.
Make sure the command '.IF' and nesting do not cross.

A2145 (E) Invalid reserved word exist in operand

The operand contains a reserved word.
Reserved words cannot be written in an operand. Rewrite the operand correctly.

A2146 (E) Symbol has already defined as another type

The symbol has already been defined in a different directive command with the same name.
You cannot define the same symbol name in directive commands '.EQU' and '.BTEQU'.
Change the symbol name.

A2147 (E) Symbol is missing

Symbol is not entered.
Write a symbol name.

A2148 (E) Invalid bit-symbol exist

An invalid bit symbol is entered.
Rewrite the bit symbol definition.

A2149 (E) Operand expression is not completed

The operand description is not complete.
Check the syntax for this operand and rewrite it correctly.

A2200 (E) No '.END' statement

'.END' is not entered.
Be sure to enter '.END' in the last line of the source program.

A2201 (E) Addressing mode specifier is not appropriate

The addressing mode specifier is written incorrectly.
Make sure that the addressing mode is written correctly.

A2202 (E) 'ALIGN' is multiple specified in '.SECTION'

Two or more ALIGN's are specified in the '.SECTION' definition line.
Delete extra ALIGN specifications.

A2203 (E) Operand value is not defined

An undefined operand value is entered.
Write a valid value for operands.

A2204 (E) Bit-symbol is in expression

A bit symbol is entered in an expression.
Bit symbols cannot be written in an expression. Check the symbol name.

A2205 (E) Invalid bit-symbol exist

An invalid bit symbol is entered.
Rewrite the bit symbol definition.

A2206 (E) The value is not constant

The value is indeterminate when assembled.
Write an expression, symbol name, or label name that will have a determinate value when assembled.

A2207 (E) Same items are multiple specified

Multiple same items of operand are specified.
Check the syntax for this operand and rewrite it correctly.

A2208 (E) Same kind items are multiple specified

Multiple operand items of the same kind are specified.
Check the syntax for this operand and rewrite it correctly.

A2209 (E) Characters exist in expression

Extra characters are written in an instruction or expression.
Check the rules to be followed when writing an expression.

A2210 (E) Format specifier is not appropriate

The format specifier is written incorrectly.
Make sure that the format specifier is written correctly.

A2211 (E) Symbol definition is not appropriate

The symbol is defined incorrectly.
Check the method for defining this symbol and rewrite it correctly.

A2212 (E) Invalid reserved word exist in operand

The operand contains a reserved word.
Reserved words cannot be written in an operand. Rewrite the operand correctly.

A2213 (E) 'JMP.S' operand label is not in the same section

Jump address for JMP.S is not specified in the same section.
JMP.S can only branch to a jump address within the same section. Rewrite the mnemonic.

A2214 (E) Reserved word is missing

No reserved word is entered.
Write a reserved word [SB], [FB], [A1], [A0], [SP], or [A1A0].

A2215 (E) No space after mnemonic or directive

The mnemonic or assemble directive command is not followed by a blank character.
Enter a blank character between the instruction and operand.

A2216 (E) No '.FB' statement

'.FB' is not entered.
When using the 8-bit displacement FB relative addressing mode, always enter '.FB' to assume a register value.

A2217 (E) No '.SB' statement

'.SB' is not entered.
When using the 8-bit displacement SB relative addressing mode, always enter '.SB' to assume a register value.

A2218 (E) No '.SECTION' statement

'.SECTION' is not entered.
Always make sure that the source program contains at least one '.SECTION'.

A2219 (E) Operand value is not defined

An undefined operand value is entered.
Write a valid value for operands.

A2220 (E) Operand size is not appropriate

The operand size is incorrect.
Check the syntax for this operand and rewrite it correctly.

A2221 (E) Operand type is not appropriate

The operand type is incorrect.
Check the syntax for this operand and rewrite it correctly.

A2222 (E) Section attribute is not defined

Section attribute is not defined. Directive command '.ALIGN' cannot be written in this section.
Make sure that directive command '.ALIGN' is written in an absolute attribute section or a relative attribute section where ALIGN is specified.

A2223 (E) Section has already determined as attribute

The attribute of this section has already been defined as relative.
Directive command '.ORG' cannot be written here.
Check the attribute of the section.

A2224 (E) Section name is missing

No section name is entered.
Write a section name in the operand.

A2225 (E) Section type is not appropriate

The section type is written incorrectly.
Rewrite the section type correctly.

A2226 (E) Section type is multiple specified

Section type is specified two or more times in the section definition line.
Only one section type CODE, DATA, or ROMDATA can be specified in a section definition line.

A2227 (E) Size or format specifier is not appropriate

The size specifier or format specifier is written incorrectly.
Rewrite the size specifier or format specifier correctly.

A2228 (E) Size specifier is missing

No size specifier is entered.
Write a size specifier.

A2229 (E) String value exist in expression

A character string is entered in the expression.
Rewrite the expression correctly.

A2230 (E) Symbol is missing

No symbol is written in the operand.
Write a symbol name in the operand.

A2231 (E) Symbol has already defined as another type

The symbol has already been defined in a different directive command with the same name.
You cannot define the same symbol name in directive commands '.EQU' and '.BTEQU'.
Change the symbol name.

A2232 (E) Symbol name is missing

The symbol name defined by '.EQU' or '.BTEQU' is not written.
Write a symbol name in the operand.

A2233 (E) Symbol was already defined as the same type

The symbol has already been defined as a bit symbol. Bit symbols cannot be redefined.
Change the symbol name.

A2234 (E) Invalid operand(s) exist in instruction

There is an invalid operand in some general instruction.
Rewrite the operand following the correct method for writing operands in a generation instruction.

A2235 (E) Syntax error in expression

The expression is written incorrectly.
Check the syntax for this expression and rewrite it correctly.

A2236 (E) Invalid operand(s) exist in instruction

There is an invalid operand in some bit instruction.
Rewrite the operand following the correct method for writing operands in a bit instruction.

A2237 (E) Operand expression is not completed

The operand description is not complete.
Check the syntax for this operand and rewrite it correctly.

A2238 (E) Too many operand

There are extra operands.
Check the syntax for these operands and rewrite them correctly.

A2239 (E) Too many operand data

There are too many operand data.
The data entered in the operand exceeds the size that can be written in one line.
Divide the instruction.

A2240 (E) Undefined symbol exist

An undefined symbol is used.
Define the symbol.

A2241 (E) Value is out of range

The value is out of range.
Write a value that matches the register bit length.

A2242 (E) Division by zero

A divide by 0 operation is attempted.
Rewrite the expression correctly.

A2243 (E) '.VER' is duplicated

'.VER' is specified more than once in the file.
'.VER' can be written only once in a file. Delete extra .VER's.

A2244 (E) '#' is missing

'#' is not entered.
Write an immediate value in this operand.

A2245 (E) ',' is missing

',' is not entered.
Insert a comma to separate between operands.

A2246 (E) ']' is missing

']' is not entered.
Write the right bracket ']' corresponding to the '['.

A2247 (E) ')' is missing

')' is not entered.
Write the right parenthesis ')' corresponding to the '('.

A2248 (E) Symbol defined by external reference data is defined as global symbol

The global symbol used here is a symbol that is defined by external reference data.
Check symbol definition and symbol name.

A2250 (E) Quote is missing

Quotes for a character string are not entered.
Enclose a character string with quotes as you write it.

A2251 (E) Right quote is missing

A right quote is not entered.
Enter the right quote.

A2252 (E) Revision information mismatch in file

Revision information of relocatable module file is different.
Confirm the version or the option of the assembler.

A2253 (E) Invalid indirect operand(s) exist in operand

The indirect addressing contains an invalid operand.
Check the syntax for this indirect addressing and rewrite it correctly.

A2254 (E) Illegal directive command is used

An illegal instruction is entered.
Rewrite the instruction correctly.

A2255 (E) '.EINSF' is missing for '.INSF'

'.EINSF', used with '.INSF' in a pair, is missing.
Check where '.INSF' is put.

A2256 (E) '.INSF' is missing for '.EINSF'

'.INSF', used with '.EINSF' in a pair, is missing.
Check where '.EINSF' is put.

A2258 (E) Invalid operand(s) exist in debug information

The debug information contains an invalid operand.
Check the syntax for this debug information and rewrite it correctly.

A2259 (E) Invalid mnemonic which isn't supported in '-R8C'

An instruction is written that cannot be used when the -R8C option is specified.
Check the written content.

A2260 (E) '.PROTECT' or '.OFSREG' is duplicated

'.PROTECT' or '.OFSREG' is specified more than once in the file.
'.PROTECT' and '.OFSREG' can be written only once in a file. Delete extra .PROTECT's or .OFSREG's.

A2261 (E) '.ID' is duplicated

'.ID' is specified more than once in the file.
'.ID' can be written only once in a file. Delete extra .ID's.

A2262 (E) Section name is not appropriate

No section name is entered.
Write a section name in the operand.

A2263 (E) Interrupt number was already defined

The software interrupt number was already defined.
Change the software interrupt number.

A2264 (E) Special page number was already defined

Special page number was already defined.
Change the special page number.

A2265 (E) Comm symbol has already defined as another type

The common symbol has already been defined in a different directive command with the same name.
Change the common symbol name.

A2266 (E) Comm symbol has already defined as differ size

The common symbol has already been defined in a different size with the same name.
Confirm a symbol size.

A2267 (E) Different register of the bank exist

A register in a different bank is written.
Check the register bank.

A2268 (E) The addressing which can't be modified is specified

The addressing which can’ t be modified is specified
Check the description rule of directive command '.INXxx'

A2269 (E) Can't use directive commands '.INXxx'

The directive command '.INXxx' cannot be used.
Check the command option.

A2270 (E) Can't use directive commands '.INXLx' or '.INXBx'

The directive command '.INXLx' or '.INXBx' cannot be used.
Check the command option.

A2271 (E) Can't use directive commands '.INXRx' or '.INXBx'

The directive command '.INXRx' or '.INXBx' cannot be used.
Check the command option.

A2272 (E) Can't use directive commands '.INXBx'

The directive command '.INXBx' cannot be used.
Check the command option.

A2273 (E) No '.LBBA' statement

'.LBBA' is not written.
When writing an instruction to specify a relative address, be sure to write '.LBBA' so that a register value will be assumed.

A2274 (E) Directive command '.RVECTOR' can't be described

The directive command '.RVECTOR' cannot be written here.
If a variable vector table is to be automatically generated, do not write a program in the vector section.

A2275 (E) Directive command '.SVECTOR' can't be described

The directive command '.SVECTOR' cannot be written here.
If a special page vector table is to be automatically generated,
do not write a program in the svector section.

A2276 (E) Invalid directive commnad which isn't supported in '-R8C'

A directive command that cannot be specified simultaneously with the -R8C option is written.
Check the content of the directive command written.

A2278 (E) Initialization function definition of the section is not appropriate

The section initialization function that involves use of C language startup is defined incorrectly.
Check how the section initialization function is defined.

A2279 (E) Invalid directive commnad '.SB_AUTO'

The directive command '.SB_AUTO' is defined incorrectly.
Check the content of the directive command written.

A2281 (E) Symbol has already defined as static type

Symbol is declared in static.
Delete directive commnad '.GLB'.

A2300 (E) Operand size is not appropriate

The operand size is incorrect.
Check the syntax for this operand and rewrite it correctly.

A2301 (E) Value is out of range

The value is out of range.
Write a value that matches the register bit length.

A2302 (E) Illegal operand is used

The operand is incorrect.
Check the syntax for this operand and rewrite it correctly.

A2303 (E) Addressing mode specifier is not appropriate

The addressing mode specifier is written incorrectly.
Make sure that the addressing mode is written correctly.

A2304 (E) Illegal directive command

An illegal instruction is entered.
Rewrite the instruction correctly.

A2305 (E) Invalid label definition

An invalid label is entered.
Rewrite the label definition.

A2306 (E) Invalid symbol definition

An invalid symbol is entered.
Rewrite the symbol definition.

A2308 (E) Questionable syntax

The structured description command is written incorrectly.
Check the syntax and write the command correctly again.

A2311 (E) ELSE not associates with IF

No corresponding IF is found for ELSE.
Check the source description.

A2312 (E) ELIF not associates with IF

No corresponding IF is found for ELIF.
Check the source description.

A2313 (E) ENDIF not associates with IF

No corresponding IF is found for ENDIF.
Check the source description.

A2314 (E) NEXT not associates with FOR

No corresponding FOR is found for NEXT.
Check the source description.

A2315 (E) WHILE not associates with DO

No corresponding DO is found for WHILE.
Check the source description.

A2316 (E) ENDS not associates with SWITCH

No corresponding SWITCH is found for ENDS.
Check the source description.

A2317 (E) BREAK' is missing for 'FOR', 'DO' or 'SWITCH'

BREAK is used in an inappropriate location.
Make sure the BREAK command is written within the FOR, DO, or SWITCH statement.

A2318 (E) 'CONTINUE' is missing for 'FOR' or 'DO'

CONTINUE is used in an inappropriate location.
Make sure the CONTINUE command is written within the FOR or DO statement.

A2320 (E) CASE not inside SWITCH

CASE is written outside a SWITCH statement.
Make sure the CASE statement is written within a SWITCH statement.

A2321 (E) DEFAULT not inside SWITCH

DEFAULT is written outside a SWITCH statement.
Make sure the DEFAULT statement is written within a SWITCH statement.

A2322 (E) Symbol is multiple defined

The symbol is defined twice or more. The macro name and some other name are duplicates.
Change the name.

A2324 (E) Undefined symbol exist

An undefined symbol is used.
Define the symbol.

A2325 (E) Division by zero

A divide by 0 operation is attempted.
Rewrite the expression correctly.

A2326 (E) DEFAULT' has already defined

There are multiple instances of DEFAULT in SWITCH.
Remove unnecessary DEFAULT statements.

A2327 (E) Section type is not appropriate

The section type is written incorrectly.
Rewrite the section type correctly.

A2328 (E) Operand value is not defined

An undefined operand value is entered.
Write a valid value for operands.

A2329 (E) Symbol has already defined as another type

The symbol has already been defined in a different directive command with the same name. You cannot define the same symbol name in directive commands ".EQU" and ".BTEQU".
Change the symbol name.

A2331 (E) No 'ENDIF' statement

No corresponding ENDIF is found for the IF statement in the source file.
Check the source description.

A2332 (E) No 'ENDS' statement

No corresponding ENDS is found for the SWITCH statement in the source file.
Check the source description.

A2333 (E) No 'NEXT' statement

No corresponding NEXT is found for the FOR statement in the source file.
Check the source description.

A2334 (E) No 'WHILE' statement

No corresponding WHILE is found for the DO statement in the source file.
Check the source description.

A2335 (E) 'CASE' has already defined as same value

The same value is written in the operands of multiple CASE statements.
Make sure the values written in the operands of CASE are unique, and not the same.

A2336 (E) Statement not preceded by 'CASE' or 'DEFAULT'

CASE or DEFAULT is preceded by a command line in the SWITCH statement.
Always be sure to write a command line after the CASE or DEFAULT statement.

A2337 (E) Symbol is missing

Symbol is not entered.
Write a symbol name.

A2338 (E) Size or Format specifier is not appropriate

The size specifier or format specifier is written incorrectly.
Rewrite the size specifier or format specifier correctly.

A3001 (F) Not enough memory

Memory is insufficient.
Divide the file and re-run. Or increase the memory capacity.

A3002 (F) Invalid option 'option' is in environment data

The environment variable contains invalid command option 'option'.
Set the environment variable correctly back again.
The options that can be set in environment variables are L, N, S, and T.

A3003 (F) Can't open file 'filename'

The 'filename' file cannot be opened.
Check the file name.

A3004 (F) Error occurred in executing 'xxx'

An error occurred when executing xxx.
Rerun xxx.

A3005 (F) Can't create Tmporary file

Temporary file cannot be generated.
Specify a directory in environment variable so that a temporary file will be created in some place other than the current directory.

A3006 (F) Illegal file name 'filename'

The file name is illegal.
Specify a file name that conforms to file name description rules.

A3007 (F) Can't find work dir

Current directory information cannot be acquired.
Execute assembling again.

A3101 (F) No input files specified

No input file is specified.
Specify a file name.

A3102 (F) Invarid option 'option' is used

An invalid command option 'option' is used.
The specified option is nonexistent. Re-input the command correctly.

A3103 (F) Ignore option 'option'

An invalid command option 'option' is specified.
The specified option is nonexistent. Input the command correctly again.

A3104 (F) Not enough memory

Memory is insufficient.
Divide the file and re-run. Or increase the memory capacity.

A3105 (F) Too many souce files

Too many files are specified.
Limit the number of files to 80 or less. Assemble the source program in several separate operations.

A3106 (F) Can't open file 'filename'

The 'filename' file cannot be opened.
Check the file name.

A3107 (F) Can't create Temporary file

Temporary file cannot be generated.
Specify a directory in environment variable so that a temporary file will be created in some place other than the current directory.

A3108 (F) Can't write file 'filename'

Data cannot be written to the 'filename' file.
Check the permission of the file.

A3109 (F) Can't create file 'filename'

The 'filename' file cannot be generated.
Check the directory capacity.

A3110 (F) Command line is too long

The command line has too many characters.
Re-input the command.

A3201 (F) Can't open file

The 'filename' file cannot be opened.
Check the file name.

A3202 (F) Can't create file

The 'filename' file cannot be generated.
Check the directory capacity.

A3203 (F) Can't read file

The 'filename' file cannot be read.
Check the permission of the file.

A3204 (F) Can't write file

The 'filename' file cannot be write.
Check the permission of the file.

A3205 (F) Illegal file name

The file name is illegal.
Specify a file name that conforms to file name description rules.

A3206 (F) Not enough memory

Memory is insufficient.
Divide the file and re-run. Or increase the memory capacity.

A3207 (F) Can't open Temporary file

The temporary file cannot be opened.
Check the directory information.

A3208 (F) Can't create Temporary file

Temporary file cannot be generated.
Specify a directory in environment variable so that a temporary file will be created in some place other than the current directory.

A3209 (F) Can't read Temporary file

The temporary file cannot read.
Check the directory information.

A3210 (F) Can't write Temporary file

The temporary file cannot be write.
Check the directory information.

A3212 (F) No 'version.txt' in environment variable LIB30

VERSION.txt cannot be found in the environment variable LIB30.
Check the environment variable LIB30.

A3213 (F) Definition of 'version.txt' is wrong

The written content of VERSION.txt present in the environment variable LIB30 is incorrect.
Delete VERSION.txt and reinstall over it.

A3304 (F) Not enough memory

Memory is insufficient.
Divide the file and re-run. Or increase the memory capacity.

A3306 (F) Can't open file 'filename'

The 'filename' file cannot be opened.
Check the file name.

A3307 (F) Can't create Temporary file 'filename'

The 'filename' file cannot be generated.
Check the directory capacity.

A3308 (F) Can't write in file 'filename'

Data cannot be written to the 'filename' file.
Check the permission of the file.

A3309 (F) Can't create file 'filename'

The 'filename' file cannot be generated.
Check the directory capacity.

A4200 (E) Internal error

An internal error occurred during processing by the assembler. Make a note of the internal error number, file name, line number, and comment in the message, and contact the support department of the vendor.

Section 12 Error Messages for the Optimizing Linkage Editor

12.1 Error Format and Error Levels

This section gives a list of error messages and explains details of errors in the following format.

Error number (Error level) **Error message**
Error details

There are five different error levels, corresponding to different degrees of seriousness.

Error Number	Error Level	Error Type	Description
L0000–L0999 P0000–P0999	(I)	Information	Processing is continued.
L1000–L1999 P1000–P1999	(W)	Warning	Processing is continued.
L2000–L2999 P2000–P2999	(E)	Error	Option analysis processing is continued; processing is interrupted.
L3000–L3999 P3000–P3999	(F)	Fatal	Processing is interrupted.
L4000– P4000–	(–)	Internal	Processing is interrupted.

12.2 Return Values for Errors

When terminating execution, each optlnk program returns a numeric value to the OS indicating its status at termination.

The table below lists the values that are returned when an error is encountered.

Return value	Content
0	Program terminated normaly. Information attributable to the file being processed occured.
1	Error, Fatal and Internal attributable to the file being processed occured. Program was forcibly terminated by input of control C.

12.3 List of Messages

L0001 (I) Section "section" created by optimization "optimization"

The section named section was created as a result of the optimization.

L0002 (I) Symbol "symbol" created by optimization "optimization"

The symbol named symbol was created as a result of the optimization.

L0003 (I) "file"-"symbol" moved to "section" by optimization

As a result of variable_access optimization, the symbol named symbol in file was moved.

L0004 (I) "file"-"symbol" deleted by optimization

As a result of symbol_delete optimization, the symbol named symbol in file was deleted.

L0005 (I) The offset value from the symbol location has been changed by optimization : file"-"section"-"symbol ± offset"

As a result of the size being changed by optimization within the range of symbol ± offset, the offset value was changed. Check that this does not cause a problem. To disable changing of the offset value, cancel the specification of the goptimize option on assembly of file.

L0100 (I) No inter-module optimization information in "file"

No inter-module optimization information was found in file. Inter-module optimization is not performed on file. To perform inter-module optimization, specify the goptimize option on compiling and assembly. Note however that the goptimize option is not available in asmsh.

L0101 (I) No stack information in "file"

No stack information was found in file. file may be an assembler output file or a SYSROF-> ELF converted file. The contents of the file will not be in the stack information file output by the optimizing linkage editor.

L0102 (I) Stack size "size" specified to the undefined symbol "symbol" in "file"

Stack size size is specified for the undefined symbol named symbol in file.

L0103 (I) Multiple stack sizes specified to the symbol "symbol"

Multiple stack sizes are specified for the symbol named symbol.

L0300 (I) Mode type "mode type 1" in "file" differ from "mode type 2"

A file with a different mode type was input.

L0400 (I) Unused symbol "file"–"symbol"

The symbol named symbol in file is not used.

L0500 (I) Generated CRC code at "address"

Generated CRC code at address.

L0510 (I) Section "section" was moved other area specified in option "cpu=<attribute>"

section without dividing is allocated according to cpu=<attribute>.

L0511 (I) Sections "section name","new section name" are Non-contiguous

section was divided and the newly created section is new section name.

L1000 (W) Option "option" ignored

The option named option is invalid, and is ignored.

L1001 (W) Option "option 1" is ineffective without option "option 2"

option 1 needs specifying option 2. option 1 is ignored.

L1002 (W) Option "option 1" cannot be combined with option "option 2"

option 1 and option 2 cannot be specified simultaneously. option 1 is ignored.

L1003 (W) Divided output file cannot be combined with option "option"

option and the option to divide the output file cannot be specified simultaneously. option is ignored. The first input file name is used as the output file name.

L1004 (W) Fatal level message cannot be changed to other level : "number"

The level of a fatal error type message cannot be changed. The specification of number is ignored. Only errors at the information/warning/error level can be changed with the change_message option.

L1005 (W) Subcommand file terminated with end option instead of exit option

There is no processing specification following the end option. Processing is done with the exit option assumed.

L1006 (W) Options following exit option ignored

All options following the exit option is ignored.

L1007 (W) Duplicate option : "option"

Duplicate specifications of option were found. Only the last specification is effective.

L1008 (W) Option "option" is effective only in cpu type "CPU type"

option is effective only in CPU type. option is ignored.

L1010 (W) Duplicate file specified in option "option" : "file"

option was used to specify the same file twice. The second specification is ignored.

L1011 (W) Duplicate module specified in option "option" : "module"

option was used to specify the same module twice. The second specification is ignored.

L1012 (W) Duplicate symbol/section specified in option "option" : "name"

option was used to specify the same symbol name or section name twice. The second specification is ignored.

L1013 (W) Duplicate number specified in option "option" : "number"

option was used to specify the same error number. Only the last specification is effective.

L1100 (W) Cannot find "name" specified in option "option"

The symbol name or section name specified in option cannot be found. The name specification is ignored.

L1101 (W) "name" in rename option conflicts between symbol and section

name specified by the rename option exists as both a section name and as a symbol name. Rename is performed for the symbol name only in this case.

L1102 (W) Symbol "symbol" redefined in option "option"

The symbol specified by option has already been defined. Processing is continued without any change.

L1103 (W) Invalid address value specified in option "option" : "address"

address specified by option is invalid. The address specification is ignored.

L1104 (W) Invalid section specified in option "option" : "section"

An invalid section is specified in "option". Observe the following:
(1) The "-output" option does not accept specification of a section that has no initial value.
(2) The "-jump_entries_for_pic" option accepts specification of only a code section and no other sections.

L1110 (W) Entry symbol "symbol" in entry option conflicts

A symbol other than symbol specified by the entry option is specified as the entry symbol on compiling or assembling. The option specification is given priority.

L1120 (W) Section address is not assigned to "section"

The "section" has no addresses specified for it. The "section" will be located at the rearmost address.
Specify the address of the section using the optlnk option "-start".

L1121 (W) Address cannot be assigned to absolute section "section" in start option

section is an absolute address section. An address assigned to an absolute address section is ignored.

L1122 (W) Section address in start option is incompatible with alignment : "section"

The address of section specified by the start option conflicts with memory boundary alignment requirements.
The section address is modified to conform to boundary alignment.

L1130 (W) Section attribute mismatch in rom option : "section 1, section 2"

The attributes and boundary alignment of section 1 and section 2 specified by the rom option are different.
The larger value is effective as the boundary alignment of section 2.

L1140 (W) Load address overflowed out of record-type in option "option"

A record type smaller than the address value was specified. The range exceeding the specified record type has been output as different record type.

L1141 (W) Cannot fill unused area from "address" with the specified value

Specified data cannot be output to addresses higher than address because the unused area size is not a multiple of the value specified by the space option.

L1150 (W) Sections in "option" option have no symbol

The section specified in "option" does not have an externally defined symbol.

L1160 (W) Undefined external symbol "symbol"

An undefined external symbol symbol was referenced.

L1170 (W) Specified SBR addresses conflict

Different SBR addresses have been specified. Processing is done with SBR=USER assumed.

L1171 (W) Least significant byte in SBR="constant" ignored

The least significant 8 bits in address constant specified by the SBR option are ignored

L1180 (W) Directive command "control directive" is duplicated in "file"

The "control directive" is written in multiple source files.
The "control directive" cannot be written more than once across files.

L1181 (W) Fail to write "type of output code"

Failed to write "type of output code" to the output file.
The output file may not contain the address to which "type of output code" should be output.
Type of output code:

    When failed to write ID code-> "ID Code"
        L1181 Fail to write "ID Code"
    When failed to write PROTECT/OFSREG code-> "Protect Code" or "OFSREG Code"
        L1181 Fail to write "Protect Code" or "OFSREG Code"
    When failed to write CRC code->"CRC Code"
        L1181 Fail to write "CRC Code"

L1182 (W) Cannot generate vector table section "section"

The input file contains vector table section. The linkage editor does not create the section automatically.

L1183 (W) Interrupt number "vector number" of "section" is defined in input file

The vector number specified by the VECTN option is defined in the input file. Processing is continued with priority given on the definition in the input file.

L1190 (W) Section "section" was moved other area specified in option "cpu=<memory attribute>"

The object size was modified through optimization of access to external variables. Accordingly, the section in the area specified by the next cpu specification was moved.

L1191 (W) Area of "FIX" is within the range of the area specified by "cpu=<memorytype>" :"<start>-<end>"

In the cpu option, the address range of <start>-<end> specified for FIX overlapped with that specified for another memory type. The setting for FIX is valid.

L1192 (W) Bss Section "section name" is not initialized

section name, which is a data section without an initial value, cannot be initialized by the initial setup program. Check the address range specified with –cpu and the sizes of pointer variables.

L1193 (W) Section "section name" specified in option "option" is ignored

option specified for the section newly created due to -cpu=stride is invalid. Do not specify option for the newly created section.

L1194 (W) Section "option" in relocation "file"-"section"-"offset" is changed.

The relocation section file offset now refers to a location in the new section created with the division of section. To prevent division, declare the contiguous_section option for section.

L1200 (W) Backed up file "file 1" into "file 2"

The file file 1 was backed up to the file file 2.

L1300 (W) No debug information in input files

There is no debugging information in the input files. The debug, sdebug, or compress option has been ignored. Check whether the relevant option was specified at compilation or assembly.

L1301 (W) No inter-module optimization information in input files

No inter-module optimization information is present in the input files. The optimize option has been ignored. Check whether the goptimize option was specified at compilation or assembly.

L1302 (W) No stack information in input files

No stack information is present in the input files. The stack option is ignored. If all input files are assembler output files or SYSROF->ELF converted files, the stack option is ignored.

L1303 (W) No rts information in input files

No information in input files to generate .rts file. The processing will end without creating an .rts file.

L1304 (W) No utl information in input files

The information necessary to generate a utl file was not input at all.

L1305 (W) Entry address in "file" conflicts : "address"

Multiple files with different entry addresses are input.

L1310 (W) "section" in "file" is not supported in this tool

An unsupported section was present in file. section has been ignored.

L1311 (W) Invalid debug information format in "file"

Debugging information in file is not dwarf2. The debugging information has been deleted.

L1320 (W) Duplicate symbol "symbol" in "file"

The symbol named symbol is duplicated. The symbol in the first file input is given priority.

L1321 (W) Entry symbol "symbol" in "file" conflicts

Multiple object files containing more than one entry symbol definition were input. Only the entry symbol in the first file input is effective.

L1322 (W) Section alignment mismatch : "section"

Sections with the same name but different boundary alignments were input. Only the largest boundary alignment specification is effective.

L1323 (W) Section attribute mismatch : "section"

Sections with the same name but different attributes were input. If they are an absolute section and relative section, the section is treated as an absolute section. If the read/write attributes mismatch, both are allowed.

L1324 (W) Symbol size mismatch : "symbol" in "file"

Common symbols or defined symbols with different sizes were input. A defined symbol is given priority. In the case of two common symbols, the symbol in the first file input is given priority.

L1325 (W) Symbol attribute mismatch : "symbol":"file"

The attribute of symbol in file does not match the attribute of the same-name symbol in other files. Check the symbol.

L1326 (W) Reserved symbol "symbol" is defined in "file"

Reserved symbol name symbol is defined in the file.

L1327 (W) Section alignment in option "aligned_section" is small : "section"

Since the boundary alignment value specified for aligned_section is 16 which is smaller than that of "section", the option settings made for that section are ignored.

L1330 (W) Cpu type "CPU type 1" in "file" differ from "CPU type 2"

Files with different CPU types were input. Processing is continued with the CPU type assumed as H8SX.

L1400 (W) Stack size overflow in register optimization

During register optimization, the stack access code exceeded the stack size limit of the compiler. The register optimization specification has been ignored.

L1401 (W) Function call nest too deep

The number of function call nesting levels is so deep that register optimization cannot be performed.

L1402 (W) Parentheses specified in option "start" with optimization

Optimization is not available when parentheses "( )" are specified in the start option. Optimization has been disabled.

L1410 (W) Cannot optimize "file"-"section" due to multi label relocation operation

A section having multiple label relocation operations cannot be optimized. Section section in file file has not been optimized.

L1420 (W) "file" is newer than "profile"

file was updated after profile. The profile information has been ignored.

L1430 (W) Cannot generate effective bls file for compiler optimization

An invalid bls file was created. This optimization is not available even if optimization of access to external variables (map option) is specified for compilation.
The optimization of access to external variables (map option) in the compiler has the following restriction. Check if this restriction is applicable and modify the section allocation.

Access to external variables cannot be optimized in some cases if a data section is allocated immediately after a code section when the base option is specified for compilation.

Note: The bls file indicates the external symbol allocation information file.
It contains the information to be used for the map option of the compiler.

L1500 (W) Cannot check stack size

There is no stack section, and so consistency of the stack size specified by the stack option on compiling cannot be checked. To check the consistency of the stack size on compiling, the goptimize option needs to be specified on compiling and assembling.

L1501 (W) Stack size overflow : "stack size"

The stack section size exceeded the stack size specified by the stack option on compiling. Either change the option used on compiling, or change the program so as to reduce the use of the stack.

L1502 (W) Stack size in "file" conflicts with that in another file

Different values for stack size are specified for multiple files. Check the options used on compiling.

L1510 (W) Input file was compiled with option "smap" and option "map" is specified at linkage

A file was compiled with smap specification. The file with smap specification should not be compiled with the map option specification in the second build processing.

P1600 (W) An error occurred during name decoding of "instance"

instance could not be decoded. The message is output using the encoding name.

L2000 (E) Invalid option : "option"
P2000 (E) Invalid option : "option"

option is not supported.

L2001 (E) Option "option" cannot be specified on command line

option cannot be specified on the command line. Specify this option in a subcommand file.

L2002 (E) Input option cannot be specified on command line

The input option was specified on the command line. Input file specification on the command line should be made without the input option.

L2003 (E) Subcommand option cannot be specified in subcommand file

The subcommand option was specified in a subcommand file. The subcommand option cannot be nested.

L2004 (E) Option "option 1" cannot be combined with option "option 2"

option 1 and option 2 cannot be specified simultaneously.

L2005 (E) Option "option" cannot be specified while processing "process"

option cannot be specified for process.

L2006 (E) Option "option 1" is ineffective without option "option 2"

option 1 requires option 2 be specified.

L2010 (E) Option "option" requires parameter

option requires a parameter to be specified.

L2011 (E) Invalid parameter specified in option "option" : "parameter"

An invalid parameter was specified for option.

L2012 (E) Invalid number specified in option "option" : "value"

An invalid value was specified for option. Check the range of valid values.

L2013 (E) Invalid address value specified in option "option" : "address"

The address address specified in option is invalid. A hexadecimal address between 0 and FFFFFFFF should be specified.

L2014 (E) Illegal symbol/section name specified in "option" : "name"

The section or symbol name specified in option uses an illegal character. Only alphanumerics, the underscore (_), and the dollar sign ($) may be used in section/symbol names (the leading character cannot be a number).

L2016 (E) Invalid alignment value specified in option "option" : "alignment value"

The alignment value specified in option is invalid. 1, 2, 4, 8, 16, or 32 should be specified.

L2017 (E) Cannot output "section" specified in option "option"

Part of the code in section specified by option cannot be output. Part of the instruction code in section has been swapped with instruction code in another section due to endian conversion. Check the section address range with respect to 4-byte boundaries in the linkage list and find which section code is swapped with the target section code.
Note: The endian conversion function is available only in the RX Family CPU.

L2020 (E) Duplicate file specified in option "option" : "file"

The same file was specified twice in option.

L2021 (E) Duplicate symbol/section specified in option "option" : "name"

The same symbol name or section name was specified twice in option.

L2022 (E) Address ranges overlap in option "option" : "address range"

Address ranges address range specified in option overlap.

L2100 (E) Invalid address specified in cpu option : "address"

An invalid address was specified in the cpu option.

L2101 (E) Invalid address specified in option "option" : "address"

The address specified in option exceeds the address range that can be specified by the cpu or the range specified by the cpu option.

L2110 (E) Section size of second parameter in rom option is not 0 : "section"

section whose size is not zero was specified in the second parameter of the rom option.

L2111 (E) Absolute section cannot be specified in rom option : "section"

An absolute address section was specified in the rom option.

L2112 (E) "section 1" and "section 2" cannot mapped as ROM/RAM in ”file”

The "section 1" and "section 2" specified in "file name" are not ROM/RAM-linked.

L2113 (E) Option "rom" and internal information in the file are conflicted

Specification of the "rom" option conflicts with the internal information.

L2120 (E) Library "file" without module name specified as input file

A library file without a module name was specified as the input file.

L2121 (E) Input file is not library file : "file (module)"

The file specified by file (module) as the input file is not a library file.

L2130 (E) Cannot find file specified in option "option" : "file"

The file specified in option could not be found.

L2131 (E) Cannot find module specified in option "option" : "module"

The module specified in option could not be found.

L2132 (E) Cannot find "name" specified in option "option"

The symbol or section specified in option does not exist.

L2133 (E) Cannot find defined symbol "name" in option "option"

The externally defined symbol specified in option does not exist.

L2140 (E) Symbol/section "name" redefined in option "option"

The symbol or section specified in option has already been defined.

L2141 (E) Module "module" redefined in option "option"

The module specified in option has already been defined.

L2142 (E) Interrupt number "vector number" of "section" has multiple definition

Vector number definition was made multiple times in vector table section. Only one address can be specified for a vector number. Check and correct the code in the source file.

L2143 (E) Invalid vector number specified: "number"

The vector number indicated by number cannot be specified.
Review the vector number specified with "#pragma special".

L2200* (E) Illegal object file : "file"

A format other than ELF format was input.
* The error number will be shown as P2200.

L2201 (E) Illegal library file : "file"

file is not a library file.

L2202 (E) Illegal cpu information file : "file"

file is not a cpu information file.

L2203 (E) Illegal profile information file : "file"

file is not a profile information file.

L2210 (E) Invalid input file type specified for option "option" : "file (type)"

When specifying option, a file (type) that cannot be processed was input.

L2211 (E) Invalid input file type specified while processing "process" : "file (type)"

A file (type) that cannot be processed was input during processing process.

L2212 (E) "option" cannot be specified for inter-module optimization information in "file"

The option option cannot be used because file includes inter-module optimization information. Do not specify the goptimize option at compilation or assembly.

L2220 (E) Illegal mode type "mode type" in "file"

A file with a different mode type was input.

L2221 (E) Section type mismatch : "section"

Sections with the same name but different attributes (whether initial values present or not) were input.

L2223 (E) Cpu type "CPU type 1" in "file" is incompatible with "CPU type 2"

A different CPU type is input.
Since these types are incompatible in part of specifications, even if the file is linked, behavior cannot be guaranteed.

L2300 (E) Duplicate symbol "symbol" in "file"

There are duplicate occurrences of symbol.

L2301 (E) Duplicate module "module" in "file"

There are duplicate occurrences of module.

L2310 (E) Undefined external symbol "symbol" referenced in "file"

An undefined symbol symbol was referenced in file.

L2311 (E) Section "section 1" cannot refer to overlaid section : "section 2"-"symbol"

A symbol defined in section 1 was referenced in section 2 that is allocated to the same address as section 1 overlaid. section 1 and section 2 must not be allocated to the same address.

L2320 (E) Section address overflowed out of range : "section"

The address of section exceeds the usable address range.

L2321 (E) Section "section 1" overlaps section "section 2"

The addresses of section 1 and section 2 overlap. Change the address specified by the start option.

L2322 (E) Section size too large: "section"

The size of section is too large. The size of a $TBR section must be 1024 bytes or less.

L2323 (E) Section "section 1 (address range)" overlaps with section "section 2 (address range)" in

hysical space
section 1 overlaps with section 2 in the physical memory. Check the addresses of the sections.
<address range>: <section start address> - <section end address>

L2330 (E) Relocation size overflow : "file"-"section"-"offset"

The result of the relocation operation exceeded the relocation size. Possible causes include inaccessibility of a branch destination, and referencing of a symbol which must be located at a specific address. Ensure that the referenced symbol at the offset position of section in the source list is placed at the correct position.

L2331 (E) Division by zero in relocation value calculation : "file"-"section"-"offset"

Division by zero occurred during a relocation operation. Check for problems in calculation of the position at offset in section in the source list.

L2332 (E) Relocation value is odd number : "file"-"section"-"offset"

The result of the relocation operation is an odd number. Check for problems in calculation of the position at offset in section in the source list.

L2340 (E) Symbol name "file"- "section" is too long

The number of characters comprising "symbol" in the "section" exceeds the translation limits of the assembler.
When you output a symbol address file, make sure the number of characters comprising the symbol name you specify does not exceed the translation limits of the assembler.

L2400 (E) Global register in "file" conflicts : "symbol", "register"

Another symbol has already been allocated to a global register specified in file.

L2401 (E) near8, near16 symbol "symbol" is outside near memory area

symbol is not allocated in the near8 or near16 range. Either change the start specification, or remove the near specifier at compilation, so that correct address calculations can be made.

L2402 (E) Number of register parameter conflicts with that in another file : "function"

Different numbers of register parameters are specified for function in multiple files.

L2403 (E) Fast interrupt register in "file" conflicts with that in another file

The register number specified for the fast interrupt general register in file does not match the settings in other files. Correct the register number to match the other settings and recompile the code.

L2404 (E) Base register "base register type" in "file" conflicts with that in another file

The register number specified for base register type in file does not match the settings in other files. Correct the register number to match the other settings and recompile the code.

L2405 (E) Option "compile option" conflicts with that in other files

Specification of "compile option" is inconsistent between the input files.
Review the compile option.

L2410 (E) Address value specified by map file differs from one after linkage as to "symbol"

The address of symbol differs between the address within the external symbol allocation information file used at compilation and the address after linkage. Check (1) to (3) below.

(1) Do not change the program before or after the map option specification at compilation.

(2) optlnk optimization may cause the sequence of the symbols after the map option specification at compilation to differ from that before the map option. Disable the map option at compilation or disable the optlnk option for optimization.

(3) When the tbr option or #pragma tbr is used, optimization by the compiler may delete symbols after the map option specification at compilation. Disable the map option at compilation or disable the tbr option or #pragma tbr.

L2411 (E) Map file in "file" conflicts with that in another file

Different external symbol allocation information files were used by the input files at compilation.

L2412 (E) Cannot open file : "file"

file (external symbol allocation information file) cannot be opened. Check whether the file name and access rights are correct.

L2413 (E) Cannot close file : "file"

file (external symbol allocation information file) cannot be closed. There may be insufficient disk space.

L2414 (E) Cannot read file : "file"

file (external symbol allocation information file) cannot be read. An empty file may have been input, or there may be insufficient disk space.

L2415 (E) Illegal map file : "file"

file (external symbol allocation information file) has an illegal format. Check whether the file name is correct.

L2416 (E) Order of functions specified by map file differs from one after linkage as to "function ame"

The sequences of a function function name and those of other functions are different between the information within the external symbol allocation information file used at compilation and the location after linkage. The address of static within the function may be different between the external symbol allocation information file and the result after linkage.

L2417 (E) Map file is not the newest version: "file name"

The .bls file is not the latest version.

L2420 (E) "file 1" overlap address "file 2" : "address"

The address specified for file 1 is the same as that specified for file 2.

P2500 (E) Cannot find library file : "file"

file specified as a library file cannot be found.

P2501 (E) "instance" has been referenced as both an explicit specialization and a generated

nstantiation
Instantiation has been requested of an instance already defined. For the file using instance, confirm that form=relocate has not been used to generate a relocatable object file.

P2502 (E) "instance" assigned to "file 1" and "file 2"

The definition of instance is duplicated in file 1 and file 2. For the file using instance, confirm that form=relocate has not been used to generate a relocatable object file.

L3000 (F) No input file

There is no input file.

L3001 (F) No module in library

There are no modules in the library.

L3002 (F) Option "option 1" is ineffective without option "option 2"

The option option 1 requires that the option option 2 be specified.

L3004 (F) Unsupported inter-module optimization information type "type" in "file"

The file contains an unsupported inter-module optimization information type. Check if the compiler and assembler versions are correct.

P3005 (F) Instantiation loop

The instance generation process is iterating in a loop.
It is possible that the input file name matches that of another file. Change the file name so that there are no matching file names except the extension.

P3007 (F) Cannot create instantiation request file "file"

Unable to create an intermediate file for the instance generation process.
Check to see if access rights of the object created folder and those beneath it are correct.

P3008 (F) Cannot change to directory "folder"

Unable to move to the "folder". Check to see if the "folder" exists.

P3009 (F) File "file"is read-only

The "file"is read-only. Change its access rights.

L3100 (F) Section address overflow out of range : "section"

The address of section exceeded FFFFFFFF. Change the address specified by the start option. For details of the address space, refer to the hardware manual of the target CPU.

L3102 (F) Section contents overlap in absolute section "section"

Data addresses overlap within an absolute address section. Modify the source program.

L3110 (F) Illegal cpu type "cpu type" in "file"

A file with a different cpu type was input.

L3111 (F) Illegal encode type "endian type" in "file"

A file with a different endian type was input.

L3112 (F) Invalid relocation type in "file"

There is an unsupported relocation type in file. Ensure the compiler and assembler versions are correct.

L3120 (F) Illegal size of the absolute code section : "section" in "file"

Absolute-addressing section in file has an illegal size. When the CPU type is RX Family in big endian, correct the size to a multiple of 4.

L3200 (F) Too many sections

The number of sections exceeded the translation limit. It may be possible to eliminate this problem by specifying multiple file output.

L3201 (F) Too many symbols

The number of symbols exceeded the translation limit. It may be possible to eliminate this problem by specifying multiple file output.

L3202 (F) Too many modules

The number of modules exceeded the translation limit. Divide the library.

L3203 (F) Reserved module name "optlnk_generates"

optlnk_generates_** (** is a value from 01 to 99) is a reserved name used by the optimizing linkage editor. It is used as an .obj or .rel file name or a module name within a library. Modify the name if it is used as a file name or a module name within a library.

L3300* (F) Cannot open file : "file"

file cannot be opened. Check whether the file name and access rights are correct.
* The error number will be shown as P3300.

L3301 (F) Cannot close file : "file"

file cannot be closed. There may be insufficient disk space.

L3302 (F) Cannot write file : "file"

Writing to file is not possible. There may be insufficient disk space.

L3303* (F) Cannot read file : "file"

file cannot be read. An empty file may have been input, or there may be insufficient disk space.
* The error number will be shown as P3303.

L3310* (F) Cannot open temporary file

A temporary file cannot be opened. Check to ensure the HLNK_TMP specification is correct, or there may be insufficient disk space.
* The error number will be shown as P3310.

L3311 (F) Cannot close temporary file

A temporary file cannot be closed. There may be insufficient disk space.

L3312 (F) Cannot write temporary file

Writing to a temporary file is not possible. There may be insufficient disk space.

L3313 (F) Cannot read temporary file

A temporary file cannot be read. There may be insufficient disk space.

L3314 (F) Cannot delete temporary file

A temporary file cannot be deleted. There may be insufficient disk space.

L3320* (F) Memory overflow

There is no more space in the usable memory within the linkage editor. Increase the amount of memory available.
* The error number will be shown as P3320.

L3400 (F) Cannot execute "load module"

load module cannot be executed. Check whether the path for load module is set correctly.

L3410 (F) Interrupt by user

An interrupt generated by (Ctrl) + C keys from a standard input terminal was detected.

L3420 (F) Error occurred in "load module"

An error occurred while executing the load module.

P3500 (F) Bad instantiation request file -- instantiation assigned to more than one file

An intermediate file for the instance generation process contains an error.
Recompile the files to be linked.

P3505 (F) Corrupted template information file or instantiation request file

An intermediate file for the template process or that for the instance generation process contains an error.
Do not edit these files.

L4000* (–) Internal error : ("internal error code") "file line number" / "comment"

An internal error occurred during processing by the optimizing linkage editor. Make a note of the internal error number, file name, line number, and comment in the message, and contact the support department of the vendor.
* The error number will be shown as P4000.

Section 13 Appendix

13.1 S-Type and HEX File Formats

This section describes the S-type files and HEX files that are output by the optimizing linkage editor.

13.1.1 S-Type File Format

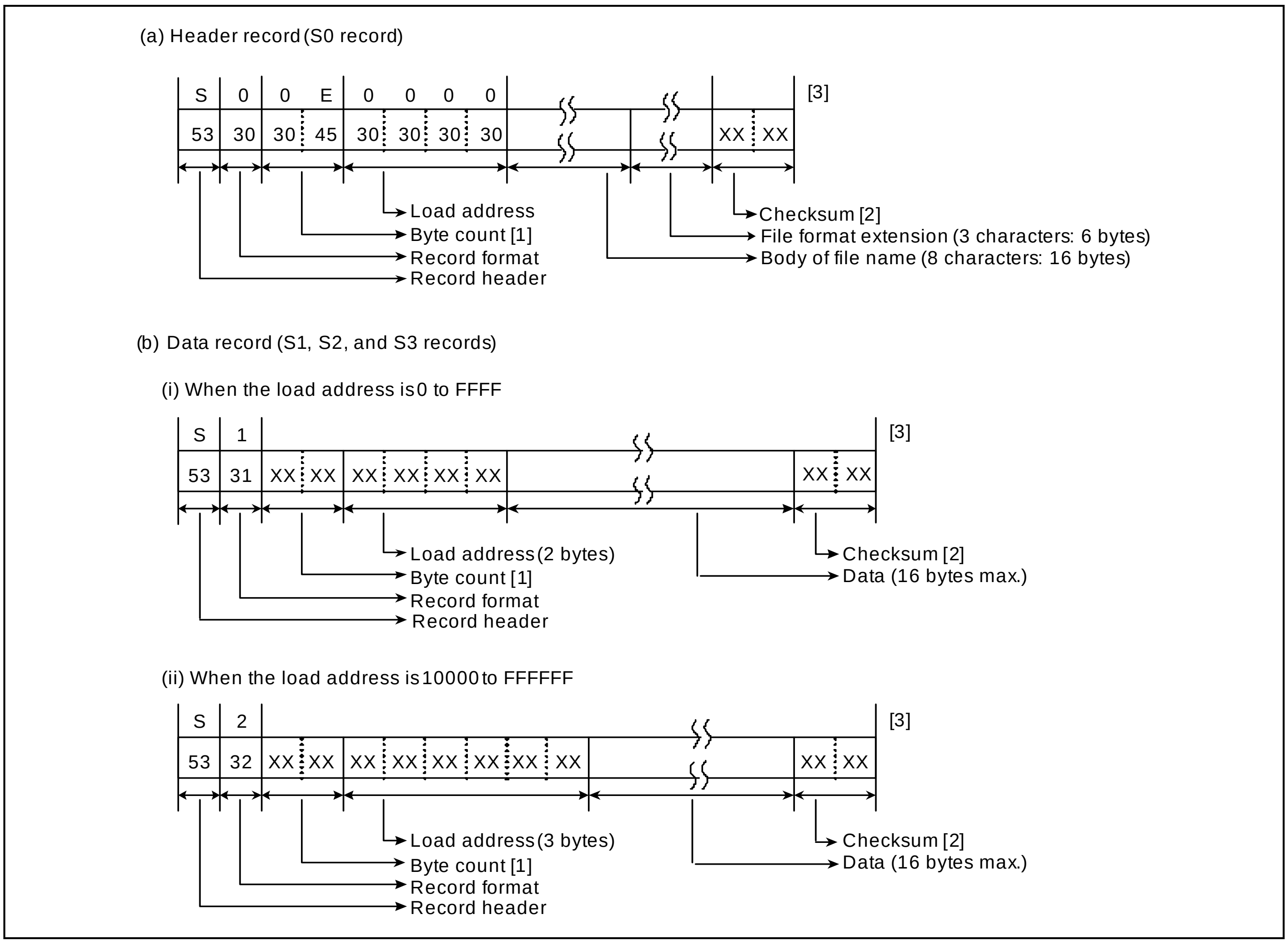
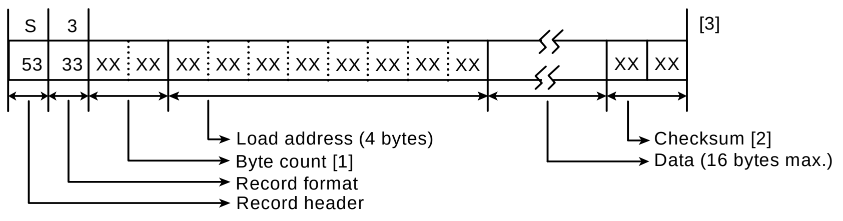

(a) Header record (S0 record)
S 0 0 E 0 0 0 0
53 30 30 45 30 30 30 30 XX XX
[3]
Load address
Byte count [1]
Record format
Record header
Checksum [2]
File format extension (3 characters: 6 bytes)
Body of file name (8 characters: 16 bytes)
(b) Data record (S1, S2, and S3 records)
(i) When the load address is 0 to FFFF
S 1
53 31 XX XX XX XX XX XX XX XX
[3]
Load address (2 bytes)
Byte count [1]
Record format
Record header
Checksum [2]
Data (16 bytes max.)
(ii) When the load address is 10000 to FFFFFF
S 2
53 32 XX XX XX XX XX XX XX XX XX XX
[3]
Load address (3 bytes)
Byte count [1]
Record format
Record header
Checksum [2]
Data (16 bytes max.)

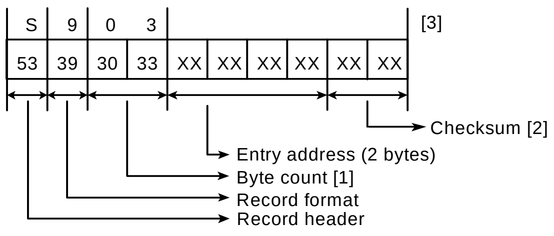
Figure 13.1 S-Type File Format

(iii) When the load address is 1000000 to FFFFFFFF

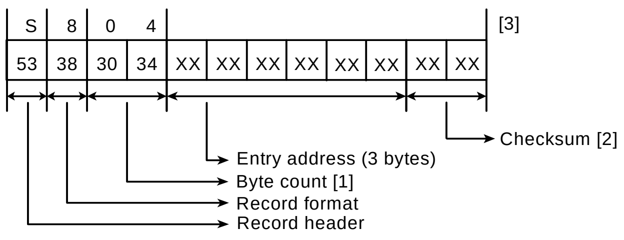


(c) End record (S9, S8, and S7 records)

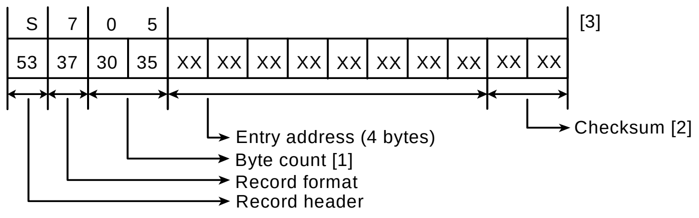
(i) When the entry address is 0 to FFFF



(ii) When the entry address is 10000 to FFFFFF



(iii) When the entry address is 1000000 to FFFFFFFF



Notes: [1] The number of bytes from the load address (or the entry address) to the checksum.
[2] 1's complement of the sum of the byte count and the data between the checksum and the byte count, in byte units.
[3] A new-line character is added immediately after the checksum.

Figure 13.1 S-Type File Format (cont)

13.1.2 HEX File Format

The execution address of each data record is obtained as described below.

- Segment address

  (Segment base address << 4) + (Address offset of the data record)

- Linear address

  (Linear base address << 16) + (Address offset of the data record)

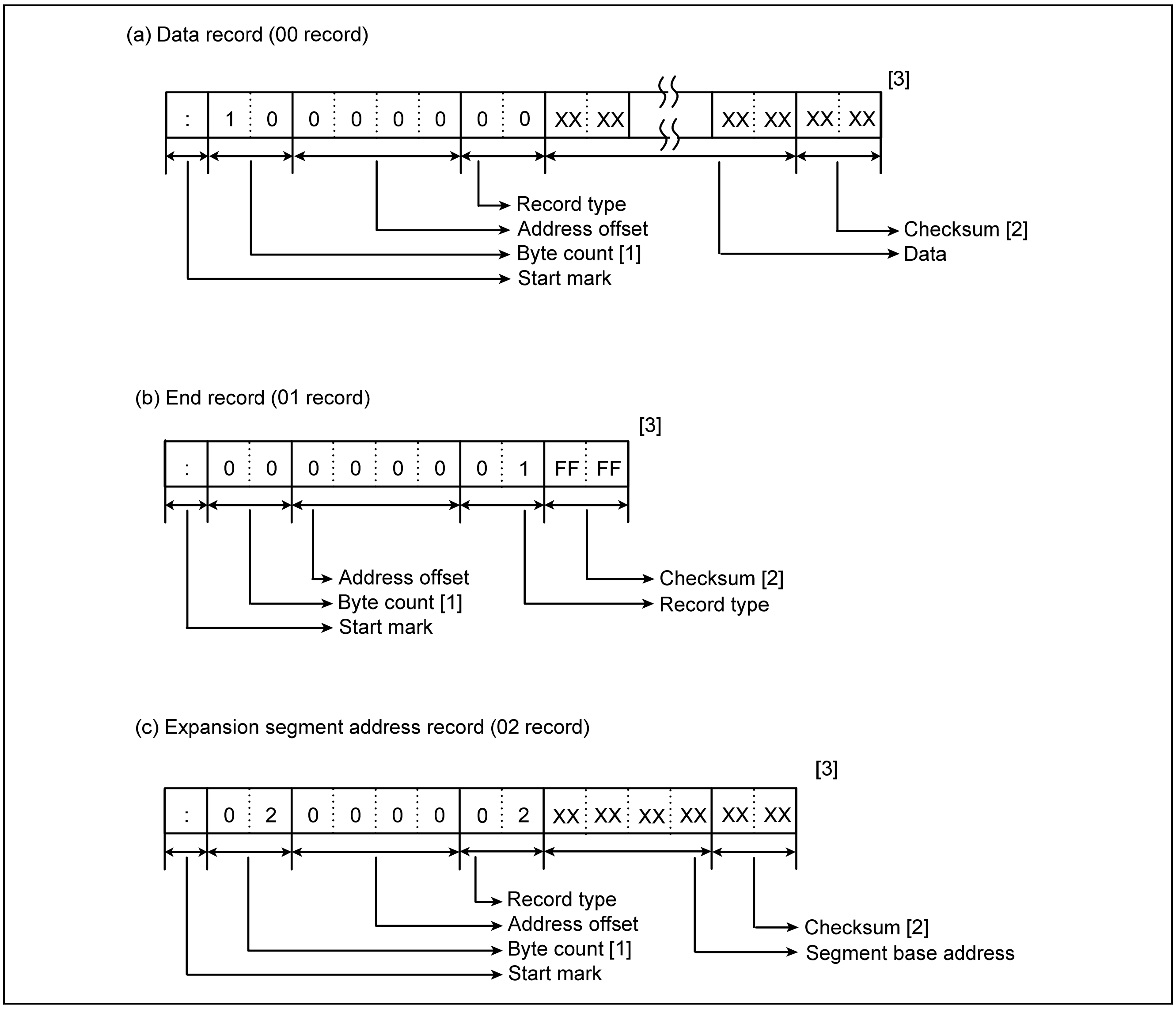
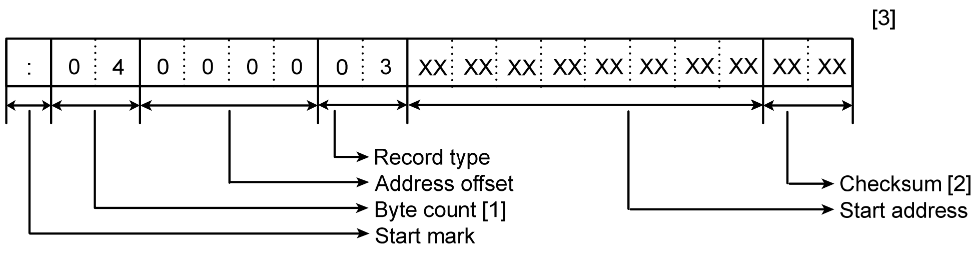
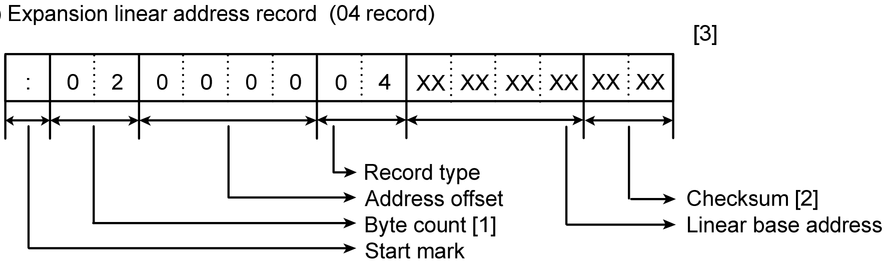


Figure 13.2 HEX File Format

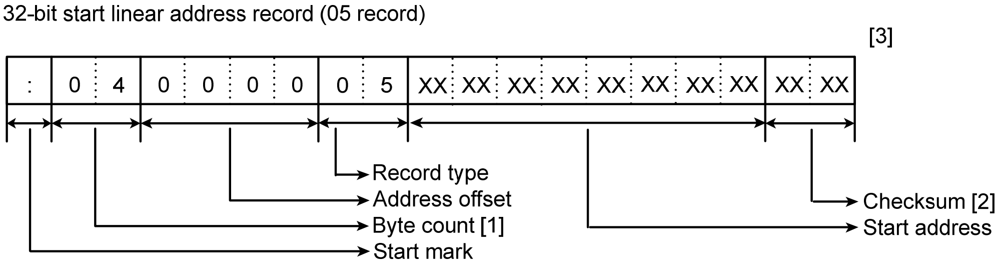
(d) Start address record (03 record)



(e) Expansion linear address record (04 record)



(f) 32-bit start linear address record (05 record)



Notes: [1] The number of bytes from the byte following the record type to the previous byte of the checksum.
[2] 2's complement of the sum of the byte count and the data between the byte count and checksum, in hexadecimal (lower 8 bits are valid).
[3] A new-line character is added immediately after the checksum.

Figure 13.2 HEX File Format (cont)

13.2 ASCII Code List

Table 16.1 ASCII Code List

Lower 4 bits	**Upper 4 bits**							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	¥	l	|
D	CR	GS	–	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

C/C++ Compiler Package for M16C Series and R8C Family V.6.00
Assembler, Optimizing Linkage Editor
User's Manual

Publication Date: Jan.16,2011 Rev.1.00

Published by: Renesas Electronics Corporation

Edited by: Renesas Solutions Corp.

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

C/C++ Compiler Package for M16C Series and R8C Family V.6.00 Assembler, Optimizing Linkage Editor User's Manual

Renesas Electronics Corporation

REJ10J2182-0100