



To all our customers

---

## **Regarding the change of names mentioned in the document, such as Mitsubishi Electric and Mitsubishi XX, to Renesas Technology Corp.**

---

The semiconductor operations of Hitachi and Mitsubishi Electric were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Mitsubishi Electric, Mitsubishi Electric Corporation, Mitsubishi Semiconductors, and other Mitsubishi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Note : Mitsubishi Electric will continue the business operations of high frequency & optical devices and power devices.

Renesas Technology Corp.  
Customer Support Dept.  
April 1, 2003

# Relocatable Assembler for M16C/80 Series

---

## AS308 V.1.00

### User's Manual

- Microsoft, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the U.S. and other countries.
- IBM, AT, and OS/2 are registered trademarks of International Business Machines Corporation.
- Intel and Pentium are registered trademarks of Intel Corporation.
- Adobe, Acrobat, and Acrobat Reader are trademarks of Adobe Systems Incorporated.
- All other brand and product names are trademarks, registered trademarks or service marks of their respective holders.

### **Precautions to be taken when using this manual**

- The information in this manual does not convey any guarantee or license for the use of the intellectual property rights or any other rights owned by Mitsubishi Electric Corporation, Mitsubishi Electric Semiconductor Systems Corporation, or third parties.
- Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation will not assume any responsibility for damage caused by or infringements on third parties' proprietary rights arising from the use of the product data, drawings, tables, programs, algorithms, or other materials in this manual.
- The product data, drawings, tables, programs, algorithms, and all other materials in this manual reflect the latest information at the time of publication. However, Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation reserve the right to make changes without notice for improvements on characteristics, etc. Please contact your nearest office of Mitsubishi or its distributor to get the latest information about the product described here before making your final decision to purchase, as necessary.
- Using technical information shown in the product data, drawings, and tables; programs, and algorithms contained in this manual, you must evaluate not only technological contents, programs, and algorithms by stand-alone, but also the overall system and judge whether they are applicable or not on the your own responsibility. Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation shall have no liability for whether they are applicable or not.
- Mitsubishi semiconductors products are neither designed nor manufactured with any intention or whatsoever to cause hindrance to social systems or a threat to human life. When you plan to use the product described here in special applications such as transport or moving vehicles, medical, aerospace, or nuclear control, or submarine repeaters or systems, please consult your nearest office of Mitsubishi or its distributor.
- This manual may not be copied, in whole or part, without the written consent of Mitsubishi Electric Corporation and Mitsubishi Electric Semiconductor Systems Corporation.

For inquiries about the contents of this manual or software, send an e-mail or fax using a text file the installer generates in the following directory to your nearest Mitsubishi office or its distributor.

\\SUPPORT\Product-name\SUPPORT.TXT

**Mitsubishi Tool Homepage** [http://www.tool-spt.mesc.co.jp/index\\_e.htm](http://www.tool-spt.mesc.co.jp/index_e.htm)

# AS308 V.1.00 User's Manual

## Contents

Specifications of AS308 .....	10
Detail of Specifications .....	11
Outline of Function .....	13
Configuration .....	13
Functions .....	15
Outline of as308 functions .....	16
Outline of ln308 functions .....	17
Outline of lmc380 functions .....	17
Outline of lb308 functions .....	17
Outline of xrf308 functions .....	17
Outline of abs308 functions .....	17
AS308 Functions .....	18
Relocatable Assemble .....	18
Unit of Address Management (Section) .....	19
Rules on Section Management .....	21
Label and symbol .....	25
Management of Label and Symbol Addresses .....	25
Library File Referencing Function .....	27
Management of Include File .....	29
Code Selection by AS308 .....	30
Optimized Selection by AS308 .....	31
Example of Optimization Selection by as308 .....	35
SB Register Offset Description .....	36
Special Page Branch .....	37
Referencing Special Page Vector Table .....	39
Macro Function .....	40
Repeat Macro Function .....	43
Conditional Assemble Control .....	43

Source Line Information Output .....	46
Symbol Definition .....	46
Environment Variables of as308 .....	46
Output messages .....	49
Compatibility with M16C/60 commands .....	51
AS308 processing when option command -mode60 is specified 52	
Replacement command list .....	53
Input/Output Files of AS308 .....	54
Relocatable Module File .....	56
Assembler List File .....	57
Assembler Error Tag File .....	58
Absolute Module File .....	59
Map File .....	60
Link Error Tag File .....	61
Motorola S Format .....	62
Intel HEX Format .....	62
Library File .....	63
Library List File .....	64
Cross Reference File .....	66
Absolute List File .....	67
Starting Up Program .....	68
Precautions on Entering Commands .....	68
Structure of Command Line .....	68
Rules for Entering Command Line .....	69
Method for Operating as308 .....	71
Command Parameters .....	71
Rules for Specifying Command Parameters .....	72
Include File Search Directory .....	74

as308 Command Options .....	74
- .....	75
-abs16 .....	76
-C .....	77
-D .....	78
-F .....	80
-H .....	81
-I .....	82
-L .....	83
-mode60 .....	84
-mode60p .....	85
-M .....	86
-N .....	87
-O .....	88
-S .....	89
-T .....	90
-V .....	91
-X .....	92
Error Messages of as308 .....	93
Warning Messages of as308 .....	108

Method for Operating In308 .....	111
Command Parameters .....	111
Rules for Specifying Command Parameters .....	112
Command File .....	113
- .....	115
-E .....	116
-G .....	117
-L .....	118
-LD .....	119
-LOC .....	120
-M .....	121
-MS/-MSL .....	122
-NOSTOP .....	123
-O .....	124
-ORDER .....	125
-T .....	126
-V .....	127
@ .....	128
Error Messages of In308 .....	129
Warning Messages of In308 .....	134
Method for Operating Imc308 .....	138
Command Parameters .....	138
Rules for Specifying Command Parameters .....	139
- .....	140
-E .....	141
-H .....	142
-ID .....	143
-L .....	144
-O .....	145
-V .....	146
-protect1 .....	147
-protect2 .....	148

Error Messages of Imc308 .....	149
Warning Messages of Imc308 .....	151
Method for Operating Ib308 .....	152
Command Parameters .....	152
Rules for Specifying Command Parameters .....	153
- .....	155
-A .....	156
-C .....	157
-D .....	158
-L .....	159
-R .....	160
-U .....	161
-V .....	162
-X .....	163
@ .....	164
Error Messages of Ib308 .....	165
Warning Messages of Ib308 .....	169
Method for Operating xrf308 .....	170
Command Parameters .....	170
Rules for Specifying Command Parameters .....	170
- .....	172
-N .....	173
-O .....	174
-V .....	175
@ .....	176
Error Messages of xrf308 .....	177
Method for Operating abs308 .....	179
Precautions using abs308 .....	179
- .....	182
-D .....	183
-O .....	184
-V .....	185



Error Messages of abs308 .....	186
Warning Messages of abs308 .....	188
Rules for Writing Program .....	189
Precautions on Writing Program .....	189
Character Set .....	190
Reserved Words .....	191
Names .....	192
Lines .....	197
Line concatenation .....	204
Operands .....	206
Rules for Writing Operands .....	206
Operators .....	209
Character String .....	212
Directive Commands .....	213
List of Directive Commands .....	214
..FILE .....	217
..MACPARA .....	219
..MACREP .....	221
.ADDR .....	223
.ALIGN .....	225
.ASSERT .....	227
.BLKA .....	229
.BLKB .....	230
.BLKD .....	231
.BLKF .....	232
.BLKL .....	233
.BLKW .....	234
.BTEQU .....	235
.BTGLB .....	237
.BYTE .....	238
.DEFINE .....	239
.DOUBLE .....	240

.ELIF .....	241
.ELSE .....	242
.END .....	243
.ENDIF .....	244
.ENDM .....	245
.ENDR .....	246
.EQU .....	247
.EXITM .....	248
.FB .....	249
.FBSYM .....	250
.FLOAT .....	251
.FORM .....	252
.GLB .....	254
.IF .....	255
.INCLUDE .....	258
.INSTR .....	259
.LEN .....	261
.LIST .....	263
.LOCAL .....	264
.LWORD .....	266
.MACRO .....	267
.MREPEAT .....	271
.OPTJ .....	273
.ORG .....	275
.PAGE .....	277
.SB .....	278
.SBBIT .....	279
.SBSYM .....	280
.SECTION .....	282
.SJMP .....	284
.SUBSTR .....	285
.VER .....	288
.WORD .....	289
? .....	291
@ .....	293

# Specifications of AS308

AS308 has been designed based on the following specifications. Make sure AS308 in your system is used within the range of this specification.

<i>Item</i>	<i>Specification</i>
Number of files opened simultaneously	Maximum 9 files
Number of characters in a file name	128 bytes (characters) on a personal computer 512 bytes (characters) on a workstation
Number of characters in a command line	128 bytes (characters) on a personal computer 512 bytes (characters) on a workstation
Number of characters that can be set in environment variables	256 bytes (characters)
Number of characters in a name	32 bytes (characters)
Total number of names	Depends on the memory capacity of the host computer by which tasks are processed.
Number of macro definitions	65535

## Detail of Specifications

### Character Set

You can use the following characters when writing an assembly program to be assembled by AS308.

#### 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

#### Lowercase 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

#### Numerals

0 1 2 3 4 5 6 7 8 9

#### Special characters

" # \$ % & ' ( ) \* + , - . / : ; [ \ ] ^ \_ | ~

#### Blank

(Space) (Tab)

#### New paragraph or line

(Carriage return) (Line feed)

- Always be sure to use 'en'-size characters when writing instructions and operands. You cannot use multi-byte characters (e.g., kanji) unless you are writing comments.

## Number of Characters on Command Line

The number of characters that can be entered on a command line is 128 characters (bytes) for the PC version or 512 characters (bytes) for the workstation version.

- The number of characters may be limited below the above specification depending on the operating environment (type of OS) of AS308.

## Method for Entering File Name

- The maximum length of a file name is 128 characters (bytes) including directory specification for the PC version or 512 characters (bytes) including directory specification for the workstation version. However, the number of characters on a command line must not exceed the above-mentioned size including the startup program name and all command options.
  - Descriptions of file names are subject to the naming conventions of the PC and workstation OS in addition to the above rules. Refer to the user's manual of your OS for details.
- Although workstations permit you to use a file name that is separated by the period (.) in two or more places, this does not apply when using AS308. AS308 allows use of the period in only one place of a file name. Furthermore, some AS308 programs restrict file name extensions (characters following the period) also. Refer to the method for starting up each AS308 program for details.

# Outline of Function

AS308 is a software system that assists you at the assembly language level in developing control programs for the M16C/80 series of single-chip microcomputers.

It converts source programs written in assembly language into files of source level debuggable format. AS308 also includes a program that converts source programs into files of M16C/80 series ROM programmable format. Furthermore, AS308 can be used in combination with the C compiler (NC308) of separate product.

## Configuration

AS308 consists of the following programs.

### Assembler driver (as308)

This program starts up macro processor and assembler processor in succession. The assembler driver can process multiple assembly source files.

### Macro processor

This program processes macro directive commands in the assembly source and performs preprocessing for the assembler processor to generate an assembly source file.

- The assembly source files generated by macro processor are erased after assembler processor finishes its processing. This does not modify the assembly source files written by the user.

### **Assembler processor**

This program converts the assembly source file preprocessed by the macro processor into a relocatable module file.

### **Linkage editor (ln308)**

This program links the relocatable module files generated by the assembler processor to generate an absolute module file.

### **Librarian (lb308)**

This program reads in a relocatable module file and generates a library file and manages it.

### **Load module converter (lmc308)**

This program converts the absolute module file generated by the linkage editor into a machine language file that can be programmed into ROM.

### **Cross referencer (xrf308)**

This program generates a cross reference file that contains definitions of various symbols and labels in the assembly source files created by the user.

### **Absolute lister (abs308)**

This program generates an absolute list file that can be output to a printer. This file is generated based on the address information in the absolute module file.

## Functions

### Relocatable programming function

This function allows you to write a program separately in multiple files. A separately written program can be assembled file by file. By allocating absolute addresses to a single file, you can debug that part of program independently of all other parts. You can also combine multiple source program files into a single debug file.

### Optimized code generation function

AS308 has a function to select the addressing mode and branch instruction that are most efficient in generating code for the source program.

### Macro function

AS308 has a macro function to improve a program's readability.

### Source level debug information output in high-level language

AS308 outputs source level debuggable high-level language information for programs developed in M16C/80 series's high-level languages.

### File generation

Each program in AS308 generates a relocatable module file, absolute module file, error tag file, list file, and others.

### IEEE-695 format file generating function

The binary files generated by AS308 are output in IEEE- 695 format. Therefore, AS308 can be shared with other M16C/80 series development tools using formats based on the IEEE-695 format.

IEEE (Institute of Electrical and Electronics Engineers, USA)



## Outline of as308 functions

- Generates relocatable module files
- Generates assembler list files

## Structure of as308

The name as308 represents a program to control these two programs.

- AS308 uses the assembler driver to control the macro processor and assembler processor. Therefore, neither macro processor nor assembler processor can be invoked directly from your command line.

## Outline of macro processor functions

- This program processes macro directive commands written in the source file.
- The processed file is available in a file format that can be processed by assembler processor.

## Outline of assembler processor functions

- This program converts the assembly languages written in the source file and those that derive from processing by macro processor into a relocatable module file.

## Outline Processing by as308

- After interpreting the input command lines, as308 activates each program of macroprocessor and assembler processor.
- as308 controls the command options added when each program starts up and the file names to be processed.
- Each program is started up sequentially in the following order:
  - 1 Macro processor
  - 2 Structured processor
  - 3 Assembler processor

## Outline of In308 functions

- Generates an absolute module file
- Generates an map file
- Assigns sections
- Utilizes relocatable modules in library file

## Outline of Imc380 functions

- Generates Motorola S format file
- Generates Intel HEX format file

## Outline of lb308 functions

- Generates a new library file
- Renewal a library file
- Generates a library list file

## Outline of xrf308 functions

- Generates a cross reference file
- Controls output information of symbol

## Outline of abs308 functions

- Generates absolute list files

# AS308 Functions

The as308 assembler converts an assembly source file into a relocatable module file that can be read in by the linkage editor. It therefore allows you to process assembly source files that include macro directive commands.

## Relocatable Assemble

- The as308 assembler is capable of relocatable assembling necessary to develop a program in separated multiple files. It generates a relocatable module file from assembly source files that contains the relocatable information necessary to link multiple files.
  - The ln308 editor references section information and global symbol information in the relocatable module file generated by as308 while it determines the addresses for the absolute module file section by section.
- Regarding hardware conditions, consider the actually used system as you write source statements and perform link processing. Hardware conditions refer to (1) RAM size and its address range and (2) ROM size and its address range.
- Programs as308 and ln308 have no concern for the physical address locations in the actual ROM and RAM of each microcomputer in the M16C/80 series. Therefore, sections of the DATA type may happen to be allocated in the chip's ROM area depending of how files are linked. When linking files, be sure to check the addresses in the actual chip to ensure that sections are allocated correctly.

## Unit of Address Management (Section)

AS308 manages addresses in units of sections.

Separation of sections are defined as follows:

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

■ Sections cannot be nested by definition.

### Start and end of a section

One section begins from a line where the section is defined and continues to the end of the source file ( the line where directive command .END is written ) or a line where another section is defined.

```

        .SECTION   ram,DATA           ; start of ram section
work:   .BLKB     10                 ; end of ram section
        .SECTION   program,CODE      ; start of program section
        JSR       sub1              ; end of program section
        .SECTION   subroutine        ; start of subroutine section
sub1:   NOP                                     ;
        MOV.W    work              ;
        RTS                                     ; end of subroutine section
        .END                               ; End of Source

```

## Type of Section

You can set a type for a section in which units addresses are controlled by AS308. The instructions that can be written in a section vary with its type.

### CODE (program area)

- This area is where a program is written.
- All commands except directive commands to allocate a memory area can be written here.
- Specify that CODE-type sections be located in a ROM area in the absolute module.

Example:

```
.SECTION    program, CODE
```

### DATA (variable data area)

- This area is where memory where contents can be modified is located.
- Directive commands to allocate a memory area can be written here.
- Specify that DATA-type sections be located in a RAM area in the absolute module.

Example:

```
.SECTION    mem, DATA
```

### ROMDATA (fixed data area)

- This area is where fixed data other than programs is written.
- Directive commands to set data can be written here.
- All commands except directive commands to allocate a memory area can be written here.
- Specify that ROMDATA-type sections be located in a ROM area in the absolute module.

Example:

```
.SECTION    const, ROMDATA
```

## Section Attributes

Attribute is assigned to a section in which units addresses are controlled by AS308 when assembling the source program.

### Relative

- Addresses in the section become relocatable values when assembled.
- The values of labels defined in a relative-attribute section are relocatable.

### Absolute

- Addresses in the section become absolute values when assembled.
- The values of labels defined in an absolute-attribute section are absolute.
- If you want to assign a section an absolute attribute, specify its address with directive command ".ORG" in a line following the line where directive command ".SECTION" is written.

Example:

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

## Alignment of sections

Relative-attribute sections can be adjusted for alignment so that their start addresses always fall on even addresses as addresses are determined when linked. If you want sections to be aligned this way, specify "ALIGN" in the operand of directive command ".SECTION".

Example:

```
.SECTION    program, CODE, ALIGN
```

## Rules on Section Management

This section describes how AS308 converts the source program written in multiple files into a single executable file.

## Section Management by as308

- Absolute-attribute section have their absolute addresses determined sequentially beginning with the specified address.
- Relative-attribute section have their (relocatable) addresses determined sequentially beginning with 0, section by section. All start addresses (relocatable) of relative-attribute sections are 0.

## Section Management by In308

- Sections of the same names in all files are arranged in order of files specified.
- Absolute addresses are determined sequentially beginning with the first section thus sorted.
- Start addresses of sections are determined sequentially beginning with 0 unless otherwise specified.
- Sections that differ each other are located at contiguous addresses unless otherwise specified.
- Sections of the same name are allocated in order of files specified.
- Sections are allocated in the order they are entered in the file that is specified first.
- If an attempt is made to allocate an absolute attribute after another absolute attribute, In308 outputs a warning.
- For section type "DATA", if addresses overlap in two or more sections, In308 outputs a warning. Sections will be allocated overlapping each other.
- For section type "CODE" or "ROMDATA", if addresses overlap in two or more sections, In308 outputs an error.
- If an attempt is made to allocate an absolute attribute after a relative attribute in sections of the same name, In308 outputs an error.
- If section names are the same and information on section attribute or type is inconsistent, In308 outputs a warning.

## Example of section allocation by In308

The following shows an example of how sections are actually allocated.

### Section definition of file1.a30

```
.SECTION  program
:
.SECTION  subroutine
.ORG 10000H
:
.END
```

### Section definition of file2.a30

```
.SECTION  subroutine
:
.SECTION  interrupt
:
.END
```

### Section definition of file3.a30

```
.SECTION  interrupt
:
.SECTION  program
:
.END
```



The example below shows how sections are located when their start addresses are not specified by a link command.

### Example 1

```
>ln308 file1 file2 file3
```

Result of section location

program	REL CODE	000000 000003	file1
	REL CODE	000003 000003	file3
subroutine	ABS CODE	001000 000003	file1
	REL CODE	001003 000002	file2
interrupt	REL CODE	001005 000002	file2
	REL CODE	001007 000003	file3

### Example 2

```
>ln308 file1 file2 file3 -order interrupt=0f000
```

Result of section location

interrupt	REL CODE	00F000 000002	file2
	REL CODE	00F002 000003	file3
program	REL CODE	00F005 000003	file1
	REL CODE	00F008 000003	file3
subroutine	ABS CODE	001000 000003	file1
	REL CODE	001003 000002	file2

## Label and symbol

The as308 assembler determines the values of labels and symbols defined in the absolute attribute section. These values are not modified even when linking. Furthermore, the label and symbol information of the following conditions are output as relocatable information:

- Global label and global symbol  
Information on global labels and global symbols are output to relocatable information.
- Local label and local symbol  
Information on local labels and local symbols are output to relocatable information providing that they are defined in the relative attribute section. However, if command options (-S and -SM) are specified when assembling, information on all local labels and local symbols are output to a relocatable file.

## Management of Label and Symbol Addresses

This section describes how the label, symbol, and bit symbol values are managed by AS308.

AS308 divides the label, symbol, and bit symbol values into global, local, relocatable, and absolute as it handles them.

The following explains the definition of each type.

### Global

- The labels and symbols specified with directive command ".GLB" are made the global labels and global symbols, respectively.
- The bit symbols specified with directive command ".BTGLB" are made the global bit symbols.
- The names defined in a file, if specified to be global, are made referencible from an external file.
- The names not defined in a file, if specified to be global, are made the external reference labels, symbols, or bit symbols that reference the names defined in an external file.

### Local

- All names specified with neither directive command ".GLB" nor ".BTGLB" are made the local names.
- Local names can be referenced within the file in which they are defined.
- Local names can have the same label name used in other files.

### Relocatable

- The values of local labels, symbols, and bit symbols in a relative-attribute section are made the relocatable values.
- The values of the externally referencible global labels, symbols, and bit symbols become relocatable values.

### Absolute

- The values of the local labels, symbols, and bit symbols defined in an absolute-attribute section become absolute values.

The diagram below shows the relationship of labels explained above.

### Converting Relocatable Values

The In308 editor converts the relocatable values in the relocatable module file into absolute values in the following manner.

- Addresses determined after relocating sections are made the absolute address.
- In the following cases, In308 outputs a warning.  
If the determined actual address lies outside the range of branch instructions and addressing modes determined by as308.

## Library File Referencing Function

If all of the following conditions are met, In308 links relocatable modules entered in a library file.

- Condition 1  
Library file reference was specified on the command line.
- Condition 2  
After all specified relocatable module files have been allocated, some global labels remain whose values are depending determination.

■ The In308 editor links the entire relocatable module where necessary global labels are defined.

### Rules for referencing library modules

The In308 editor determines the relocatable modules to be linked in the order described below. A relocatable module that has been determined to be linked is relocated section by section. Sections are relocated in the same way as sections are relocated in a relocatable module file.

- 1 The In308 first searches the global label information of relocatable modules entered in a library file. Relocatable modules are referenced in the order they are entered in the library file.
- 2 The labels searched from the library file are compared with the labels whose values are pending. If any labels match, In308 links this relocatable module in the library file to the absolute module file.
- 3 After going over the relocatable modules in the library file, if there remains any global label whose value is pending (i.e., a relocatable module in the library file contains an external reference label), In308 again searches modules in the library file in the order they are entered.

## Example of referencing library modules

libsmp1.a30

```
.GLB      sym1      ; External reference symbol
.SECTION  progr1
:
.END
```

libsmp.lib

libsmp2.a30

```
sym2      .GLB      sym2
           .EQU      2
           .SECTION  prog2
:
           .END
```

libsmp3.a30

```
sym1      .GLB      sym1
           .GLB      sym2 ; External reference symbol
           .EQU      1
           .SECTION  prog3
:
           .END
```

Input command to link

```
>ln308 libsmp1 -L libsmp.lib
```

Result to link

```
processing "libsmp1.r30"
processing "libsmp.lib ( libsmp3.r30 )"
processing "libsmp.lib ( libsmp2.r30 )"
```

## Management of Include File

The as308 assembler can read an include file into any desired line of the source program. This facility can be used to improve the legibility of your program.

### Rules for Writing Include File

To write an include file, follow the same rules that you follow for writing a source program.

- Directive command ".END" cannot be written in an include file.

### Reading Include File into Source Program

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

Example:

```
.INCLUDE    initial.inc
```

## Code Selection by AS308

The as308 assembler is designed to choose the shortest code possible from the M16C/80 series's addressing modes. This section outlines the M16C/80 series's addressing modes and explains how to write mnemonics in the source program.

### Optimized Code Selection

The as308 assembler optimizes code selection when one of the following conditions applies:

- Operands that have a valid value when assembling in which however, no addressing mode is specified
- Operands in which symbols declared in ".SBSYM" or ".FBSYM" are used.

### Outline of Mnemonic Description

The M16C/80 series allows you to write the specifiers listed below and an addressing mode in its mnemonics and operands. The specifiers and addressing modes you can specify differ with each mnemonic. Refer to the "M16C/80 Series Software Manual" for details on how to write mnemonics.

- **Size specifier**  
Specify the size of the data to be operated on by the mnemonic. You cannot omit this specifier; it must always be entered.
- **Jump distance specifier**  
Specify the distance to the jump address of a branch instruction or subroutine call instruction. (You normally do not need to specify this.)
- **Instruction format specifier**  
Specify the format of op-code. The code lengths of op-code and operand differ with each op-code format. (You normally do not need to specify this.)
- **Addressing mode**  
Specify the addressing mode of operand data. You can omit this specification. The section to specify the address range of relative addressing in AS308 is referred to as an addressing mode specifier.
- Here, ':16' and ':8' are the addressing mode specifiers.

```
MOV.W      work1:16[SB],work2:8[SB]
```

## Optimized Selection by AS308

The as308 assembler generates optimum-selected or most suitable code for the source statements shown below.

- When jump distance specifier is omitted
  - The jump distance specifier cannot be omitted if the operand is indirect addressing. An error is generated if this specifier is omitted.
  - When instruction format specifier is omitted
  - When addressing mode specifier is omitted
  - For an addressing mode with displacement, be sure to specify the displacement.
  - Combination of the above
- The following explains optimum selection by as308 for each case listed above.

### When jump distance specifier is omitted (normally omitted)

The as308 assembler performs optimum selection when all of the following conditions are met:

- When the operand is written with one label.
- When the operand is written with an expression that contains one label.  
Label + value determined when assembled  
Label - value determined when assembled  
Value determined when assembled + label
- When operand labels are defined in the same section.
- The section where the instruction is written and the section where the operand label is defined both are absolute-attribute sections and are written in the same file.
- If conditions to perform optimum selection are not met, as308 generates code as directed by directive command ".OPTJ".



The following shows instructions selected by as308.

- Unconditional branch instruction  
The shortest instruction possible to branch is selected from jump distances '.A', '.W', '.B', and '.S'.
- Size '.S' is selected only when the branch instruction and the jump address label are present in the same section.
- Subroutine call instruction  
The shortest instruction possible to branch is selected from jump distances '.A' and '.W'.
- Conditional branch instruction  
Jump distance '.B' or alternative instruction is generated.
- The source line information in a list file is output directly as written in the source lines. Code of alternative instruction is output to the code information section.
- Optimum selection is not performed for the 'ADJNZ' instruction.

### **When instruction format specifier is omitted (normally omitted)**

The instruction format specifier normally is omitted.

The as308 assembler performs optimum selection for mnemonics where instruction format specifiers are omitted.

If instruction format specifiers are omitted, as308 first determines the addressing mode before it selects the instruction format.

### When addressing mode specifier is omitted

If addressing mode specifiers are omitted, as308 selects the most suitable code in the following manner:

- In cases of addressing with displacement, if the displacement value is determined when assembled, the most suitable addressing mode is selected.
- If directive command ".SB" or ".FB" is defined, an 8-bit SB relative addressing mode (hereafter called SB relative) or 8-bit FB relative addressing mode (hereafter called FB relative) is selected depending on condition.

The next page shows the condition under which one of the two addressing modes above is selected.

### Selection of SB relative

SB relative is selected when the following conditions are met.

- The SB register value must always be set using directive command ".SB" before SB relative addressing can be used.
- When an operand value is determined when assembling the source program and the determined value is in an addressing range in which SB relative can be selected.  
The SB relative selectable address range is a range in the 64-Kbyte address space and range in the result added -0 to +255 to value of the 16-bit register (SB).
- Optimization is not performed unless the SB register value is defined by an expression in which it will be determined when assembling the source program.
- When the symbol declared by directive command ".SBSYM" is written in the op-code.

**For 1-bit operation instructions, the addressing mode is selected in the following manner:**

- When the mnemonic has a short format in its instruction format...  
Short format SB relative is selected.
- When the mnemonic does not have a short format in its instruction format...  
A 16-bit SB relative addressing mode is selected.

**Selection of FB relative**

FB relative is selected when the following conditions are met.

- The FB register value must always be set using directive command ".FB" before FB relative addressing can be used.
- When an operand value is determined when assembling the source program and the determined value is in an addressing range in which FB relative can be selected.  
This address range is a range in the 64-Kbyte address space and range in the result added -128 to +127 to value of the 16-bit register (FB).
- When the symbol declared by directive command ".FBSYM" is written in the op-code.
- When the following expression that includes a symbol defined by directive command ".FBSYM" is written in the op-code.
  - (symbol) - value determined when assembled
  - (symbol) + value determined when assembled
  - Value determined when assembled + (symbol)

## Example of Optimization Selection by as308

The examples below show the addressing modes optimum selected by as308 and how they are written in the source file.

### Address register relative with 8-bit displacement

Example:

```
sym1 .EQU      11H
ABS.B sym1+1[A0]
```

### SB relative

Example 1:

```
sym2 .EQU 2
sym3 .EQU 3
.SB      0
.SBSYM   sym3
ABS.B sym3-sym2
```

Example 2:

```
.SB      100H
sym4 .EQU 108H
ABS.B sym4
```

## SB Register Offset Description

Programming with AS308 allows you to enter a description to specify an offset address from the SB register value.

### Function

- Operation is performed on the address value specified by the directive command ".SB" plus a specified offset value.
- Code is generated in SB relative addressing mode.

### Rules for writing command

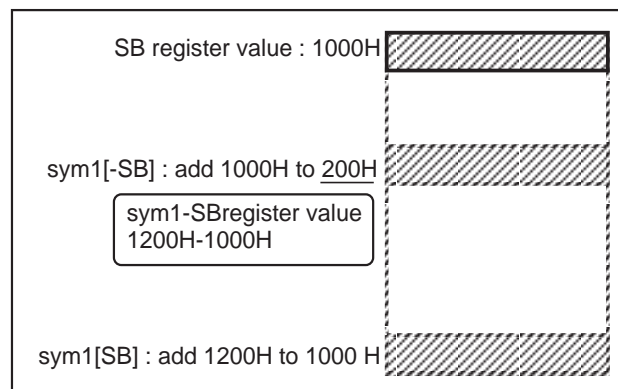
- This description can be entered for an operand where the SB relative addressing mode can be written.
- A label, symbol, or numeric value can be used to write the offset.

### Description example

```

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

```



## Special Page Branch

The M16C/80 series assembly language allows you to branch at a special page using a special page vector table by writing a "JMPS" mnemonic.

## Special Page Subroutine

The M16C/80 series assembly language allows you to call a special page subroutine using a special page vector table by writing a "JSRS" mnemonic.

## Special Page Vector Table

The following outlines the special page vector table:

- The special page vector table is allocated in addresses 0FFFE00H to 0FFFFDBH.
- One vector table consists of two bytes.
- Each vector table is assigned a special page number.
- The special page number decreases from 255 to 254, and so on every 2 bytes beginning with address 0FFFE00H.

For details about the special page vector table, refer to the "M16C/80 Series Software Manual."

This manual only shows how to set and reference the special page vector table.

## Setting Special Page Vector Table

The special page vector table is used to store the 16 low-order bits of an address in the special page.

### Rules for writing command

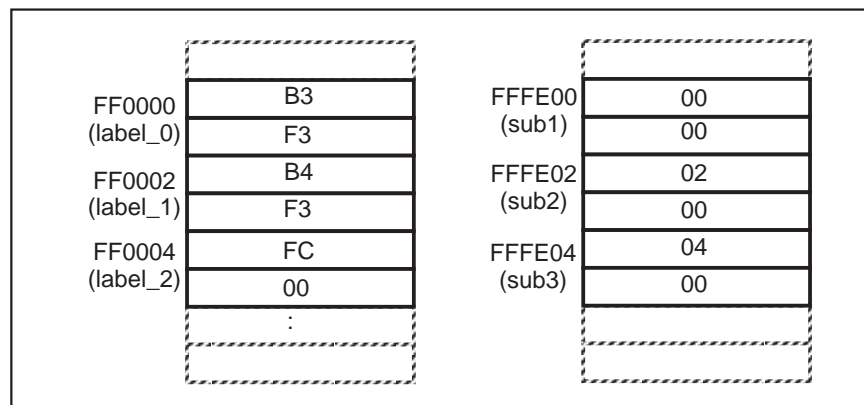
- Always be sure to define a section.
- Use the directive command ".ORG" to define the absolute address.
- The address you set here must be an even-numbered address.
- Use the directive command ".WORD" to store the 16 low-order bits of an address in the special page in ROM.

### Description example

```

        .SECTION      sp_vect,ROMDATA
        .ORG          0FFFE00H
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          0FFFFDAH
sub238: .WORD        label_237 & 0FFFFH

```



## Referencing Special Page Vector Table

There are two methods to reference the special page vector table as described below.

- Specify your desired special page number.
- Specify the address of your desired special page vector table.

### Rules for writing command

- When specifying a special page number, always be sure to write "#" at the beginning of the number.
- When specifying the address of a special page vector table, always be sure to write "\" at the beginning of the address.

### Description example

```
.SECTION    p
main:
    JSRS    \sub1
    JSRS    \sub2
    JSRS    \sub3
.SECTION    special
.ORG    0FF0000H
label_0:
    MOV.B  #0,R0H
    RTS
label_1:
    MOV.B  #0,R0L
    RTS
label_2:
    JMP    main
.END
```

The content of ".SECTION p" in the above example can be written differently, like the one shown below.

```
.SECTION    p
main:
    JSRS    #255
    JSRS    #254
    JMPS   #253
```



## Macro Function

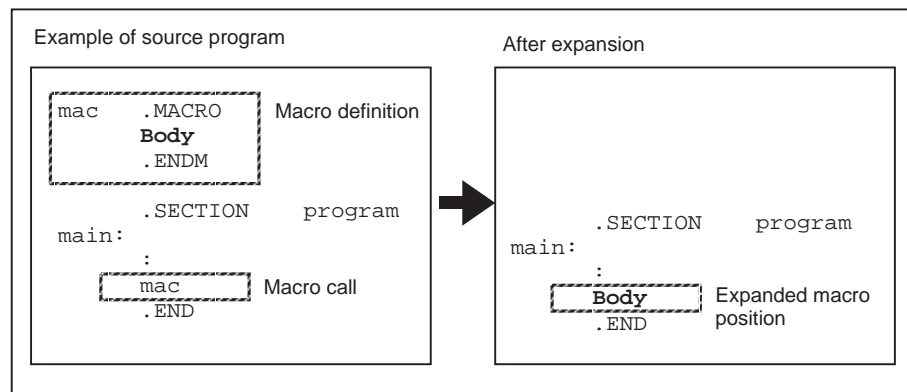
This section describes the macro functions that can be used with AS308. The following lists the macro functions available for AS308:

- **Macro function**  
To use a macro function, define it with directive commands ".MACRO" and ".ENDM" and call up the defined macro.
- **Repeat macro function**  
To use a repeat macro function, use directive commands ".MREPEAT" and ".ENDR" to define it.

Each macro function is described below.

### Macro Function

- A macro function can be used by defining a macro name (macro definition) and calling up the macro (macro call).
- A macro function cannot be made available for use by macro definition alone.
- Macro definition and macro call have the following relationship.



## Macro Definition

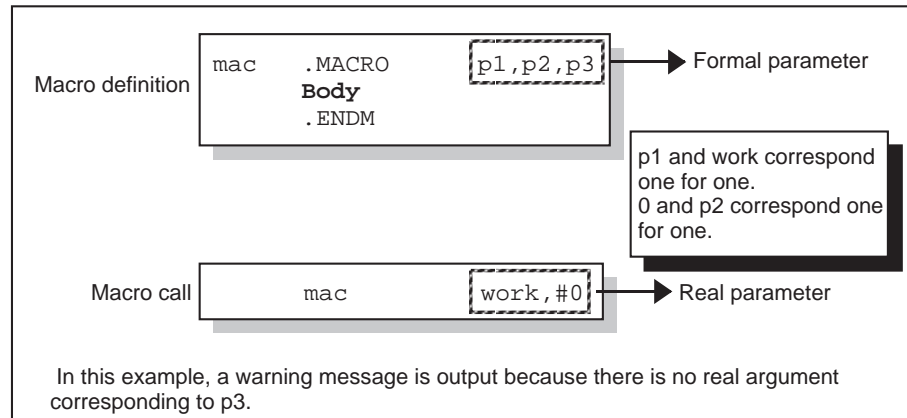
- Macro definition means defining a collection of more than one line of instructions to a single macro name by using directive command ".MACRO".
  - Macro names and macro arguments are case-sensitive, so that lowercase and uppercase letters are handled differently.
  - End of macro definition is indicated by directive command ".ENDM".
  - Lines enclosed with directive commands ".MACRO" and ".ENDM" are called the macro body.
  - Macro definition can have formal parameters defined.
  - Recursive definition is allowed for macro definition.
  - Macros can be nested in up to 65,535 levels including both macro definition and macro call.
  - Macros of the same name can be redefined.
  - Macro definition can be entered outside the range of a section.
  - Any instructions you can write in source programs can be written in the macro body.
- Bit symbols cannot be written in the macro body.
- Macro formal parameters (up to 80 arguments) can be written.
  - Macro local labels can be written for up to a total of 65,535 labels in one assembly source file.

## Macro Local Labels


- The labels defined with directive command ".LOCAL" are made macro local labels.
- Macro local labels can be used in only macro definition.
- The label names declared to be macro local labels are allowed to be written in places outside the macro with the same name.
- The labels you want to be used as macro local labels must first be declared to be a macro local label before you define the label.

## Macro Call

- Macro call can be accomplished by writing a macro name that has been defined with directive command ".MACRO".
- Code for the macro body is generated by macro call.
- Macro names cannot be forward referenced (i.e., you cannot call a macro name that is defined somewhere after the line where macro call is written). Always make sure that macro definition is written in places before the macro call line.
- Macro names cannot be externally referenced (i.e., you cannot call a macro name that is defined in some other file). If you want to call the same macro from multiple files, define the macro in an include file and include it in your source file.
- The content of the macro-defined macro body is expanded into the line from which the macro is called.
- Actual parameters corresponding to macro-defined formal parameters can be written.



## Repeat Macro Function

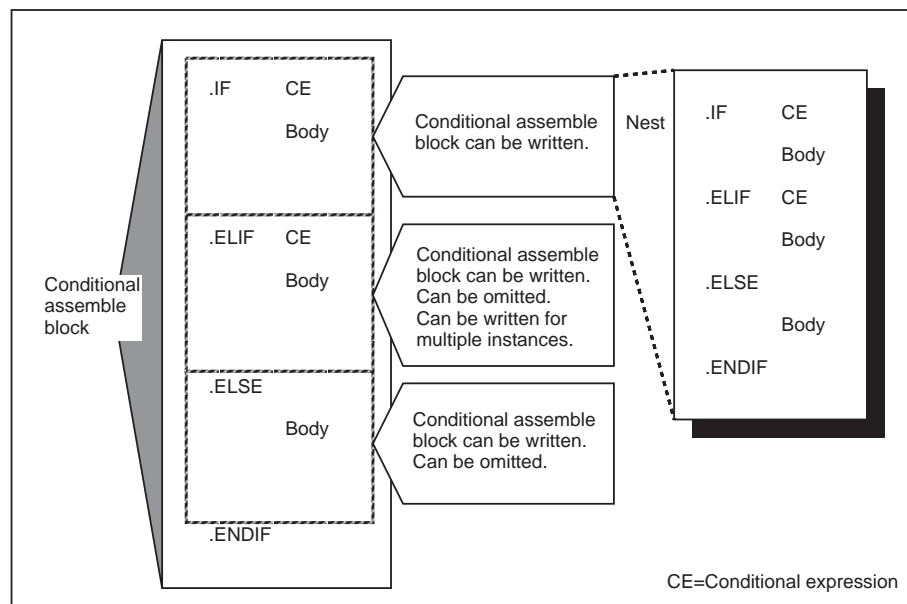
- The macro body enclosed between directive commands ".MREPEAT" and ".ENDR" is expanded repeatedly into places after the specified line a specified number of times.
  - Repeat macros are expanded into the defined line.
  - Labels can be written in repeat macro definition lines.
-  This label is not a macro name. There is no macro call available for repeat macros.

## Conditional Assemble Control

The as308 assembler allows you to specify whether or not you want a specified range of lines to be assembled by using conditional assemble directive commands.

### Configuration of Conditional Assemble Block

The diagram below shows the configuration of a conditional assemble block.



## Executing Conditional Assemble

The following shows examples of how conditional assemble is executed after selecting from three messages. Here, the assembly source file name is "sample.a30".

Conditional assemble execution examples are shown below.

Asseby Source File)

```
.SECTION    outdata,ROMDATA,ALIGN
. IF    TYPE==0
. BYTE "PROTO TYPE"
. ELIF TYPE>0
. BYTE "MASS PRODUCTION TYPE"
. ELSE
. BYTE "DEBUG MODE"
. ENDIF
. END
```

Command Input 1)

```
>as308 sample -Dtype=0
```

Assembled Result 1)

```
.SECTION    outdata,ROMDATA,ALIGN
. BYTE "PROTO TYPE"
. END
```

## Command Input 2)

```
>as308 sample -Dtype=1
```

## Assembled Result 2)

```
.SECTION  outdata,ROMDATA,ALIGN
.BYTE     "MASS PRODUCTION TYPE"
.END
```

## Command Input 3)

```
>as308 sample -Dtype=-1
```

## Assembled Result 3)

```
.SECTION  outdata,ROMDATA,ALIGN
.BYTE     "DEBUG MODE"
.END
```

Next, the following shows an example of how to set a value to "TYPE" in the assembly source file.

```
TYPE      .EQU      0
          .SECTION  outdata,ROMDATA,ALIGN
          .IF      TYPE==0
          .BYTE    "PROTO TYPE"
          .ELIF    TYPE>0
          .BYTE    "MASS PRODUCTION TYPE"
          .ELSE
          .BYTE    "DEBUG MODE"
          .ENDIF
          .END
```

## Source Line Information Output

The as308 assembler outputs to a relocatable module file the information that is necessary to implement source debugging of "NC308" and "Macro description of AS308."

## Symbol Definition

The as308 assembler allows you to define symbols by entering command option (-D) when starting up the program. This function can be used in combination with a condition assemble function, etc.

## Environment Variables of as308

The as308 assembler references the environment variables listed below.

<i>Environment Variables</i>	<i>Program</i>
AS308COM	as308
BIN308	as308
INC308	as308
LIB308	ln308
TMP308	as308,ln308,lb308

## AS308COM

The as308 assembler adds the command options set in the environment variables as it processes a file.

The command options set in this environment variable can be nullified by using two consecutive hyphens (--).

The command options listed below can be set to this environment variable.

## How to set up AS308COM

PC version

```
SET AS308COM=-L -N -S -T
```

Workstation version

```
setenv AS308COM '-L -N -S -T'
```

- When setting a character string containing spaces to this environment variable while operating on a workstation, always be sure to enclose the character string with quotations as you enter it.

## How to clear the settings on AS308COM

PC version

```
SET AS308COM=
```

Workstation version

```
unsetenv AS308COM
```

## Example for using AS308COM

When environment variable AS308COM is set, as308 sets the command options in the following order.

- 1 as308 first sets the command options set in AS308COM.
- 2 as308 sets the command options entered from a command line.

The following shows an example for setting an option to AS308COM, an example for entering a command option from a command line, and an example of a valid command option.

## Example for setting up AS308COM

```
SET OPT308=-L -N -S -L
```

Command input example -1

```
as308 -Dsym=0 --N
```

Option that becomes valid when executing as308 -1

```
as308 -Dsym=0 -L -S -T
```

Command input example -2

```
as308 -O\tmp --T -SM -LM
```

Option that becomes valid when executing as308 -2

```
as308 -O\tmp -N -SM -LM
```



## **BIN308**

The assembler driver (as308) invokes the macro processor and assembler processor residing in the directory you have set.

If this variable is set, always make sure that macroprocessor and assembler processor are placed in the directory you have set.

## **INC308**

The assembler as308 retrieves include files written in the assembly source file from the directory set in INC308. Multiple directories can be specified. If multiple directories are specified, as308 searches the directories sequentially from left to right in the order they are written.

### **How to set INC308**

Personal computer version

Separate the directory names with a semicolon as you write them.

```
SET INC308=C:\COMMON;C:\PROJECT
```

Workstation version

Separate the directory names with a colon as you write them.

```
setenv INC308 /usr/common/:/usr/project
```

## **TMP308**

Programs generate a work file necessary to process files in the directory that is set in this environment variable.

The work file normally is erased after as308 finishes its processing.

## Output messages

The programs are included this products output informations of process to screen.


### Error Messages

This chapter describes the error messages output by each AS308 program.

### Types of Errors

There are following two types of error messages.

- **Error message**  
This refers to an error encountered during program execution that renders the program unable to perform its basic function.
- **Warning message**  
This refers to an error encountered during program execution that presents some problem even though it is possible to perform the basic function of the program.

 Please try to solve all problems that have caused generation of a warning message. Some warnings may result in a fault when operating your system on the actual chip although no problem might have been encountered during debugging.

## Return Values for Errors

When terminating execution, each AS308 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.

<i>Return value</i>	<i>Content</i>
0	Program terminated normally.
1	Program was forcibly terminated by input of control C.
2	Error relating to the OS's file system or memory system occurred.
3	Error attributable to the file being processed occurred.
4	Error in input from the command line occurred.

## Compatibility with M16C/60 commands

AS308 supports the `-mode60` command option. This option assembles programs developed with AS30.

■ The below commands cannot be replaced even when using the `-mode60` command option. Change the source program.

- MOVA src,R0
- MOVA src,R1
- MOVA src,R2
- MOVA src,R3
- src= dsp[A0],dsp[A1],dsp[SB],dsp[FB],abs16
- JMPL.A A1A0
- JSRL.A A1A0
- PUSHC INTBL
- PUSHC INTBH
- POPC INTBL
- POPC INTBH
- MUL.W generic,A0
- MULU.W generic,A0
- generic= R0,R1,R2,R3,A0,A1,[A0],[A1]  
dsp[SB],dsp[FB],dsp[A0],dsp[A1],abs16
- LDC
- STC
- LDE.B/W [A1A0],generic
- STE.B/W generic,[A1A0]
- generic= R0L/R0,R0H/R1,R1L/R2,R1H/R3  
A0,A1,[A0],[A1],dsp[SB],dsp[FB]  
dsp[A0],dsp[A1],abs16
- BSET:G bit,R2
- BSET:G bit,R3
- BAND, BCLR, BNAND, BNOR, BNOT, BNTST, BNxor, BOR, BTST, BTSTC, BTSTS, BXOR and BM are not replaced in the same way as BSET.

## AS308 processing when option command -mode60 is specified

When the command option -mode60 is specified, processing with AS308 is as follows.

- Format specifiers of the MOV, CMP, OR, SUB, AND, NOT, PUSH and POP commands are ignored.
- Addressing mode specifiers of the JMPI and JSRI commands are ignored.
- Format specifiers of the ADD command are ignored.
- When adding to the stack pointer (SP) using the ADD command, the size specifier is processed as ".L".
- LDINTB is replaced with LDC before processing.
- The STZ, STNZ and STZX commands are processed by byte size.
- The LDE and STE commands are replaced with the MOV command before processing.
- 1-bit operating commands are replaced with commands for AS308 before processing.
- Bit operating commands BAND, BCLR, BNAND, BNOR, BNOT, BNTST, BNXOR, BOR, BTST, BTSTC, BTSTS, BXOR and BMcmd are replaced in the same way as BSET. See the replacement command list.

## Replacement command list

<i>AS30 source format</i>	>>	<i>Format when replaced</i>
LDINTB #imm20		LDC #imm24,INTB
LDE.B/W abs:20,dest		MOV.B/W abs,dest
LDE.B/W dsp:20[A0],dest		MOV.B/W dsp[A0],dest
STE.B/W src,abs:20		MOV.B/W src,abs
STE.B/W src,dsp:20[A0]		MOV.B/W src,dsp[A0]
BSET:G bit,R0		BSET bit,R0L BSET bit,R0H
BSET:G bit,R1		BSET bit,R1L BSET bit,R1H
BSET:G bit,A0		Assembly is possible for bits 0 ~ 7.
BSET:G bit,A1		Assembly is possible for bits 0 ~ 7.
BSET:G [A0]		BITINDEX.B [A0] BSET 0,0
BSET:G [A1]		BITINDEX.B [A1] BSET 0,0
BSET:G base:8[A0]		BITINDEX.B [A0] BSET 0,base
BSET:G base:16[A0]		BITINDEX.B [A0] BSET 0,base
BSET:G base:8[A1]		BITINDEX.B [A1] BSET 0,base
BSET:G base:16[A1]		BITINDEX.B [A1] BSET 0,base
BSET:G bit,base:8[SB]		BSET bit,base[SB]
BSET:S bit,base:11[SB]		
BSET:G bit,base:16[SB]		
BSET:G bit,base:8[FB]		BSET bit,base[FB]
BSET:G bit,base:16		BSET bit,base

# Input/Output Files of AS308

The table below lists the types of files input for AS308 and those output by AS308. Any desired file names can be assigned. However, if the extension of a file name is omitted, AS308 adds the extension shown in ( ) in the table below by default.

## as308

<i>Input Files</i>	<i>Output Files</i>
Source file (.a30)	Relocatable module file (.r30)
Include file (.inc)	Assembler list file (.lst)
	Assembler error tag file (.atg)

## In308

<i>Input Files</i>	<i>Output Files</i>
Relocatable module file (.r30)	Absolute module file (.x30)
Library file (.lib)	Map file (.map)
	Linkage error tag file (.ltg)

## Imc308

<i>Input Files</i>	<i>Output Files</i>
Absolute module file (.x30)	Motorola S format file (.mot)
	Intel HEX format file (.hex)

**lb308**

<i>Input Files</i>	<i>Output Files</i>
Relocatable module file (.r30)	Library file (.lib) Relocatable module file (.r30) Library list file (.lls)

**xrf308**

<i>Input Files</i>	<i>Output Files</i>
Source file (.a30) Assembler list file (.lst)	Cross reference file (.xrf)

**abs308**

<i>Input Files</i>	<i>Output Files</i>
Absolute module file (.x30) Assembler list file (.lst)	Absolute list file (.als)



## Relocatable Module File

A relocatable module file is one of the files generated by as308. This file is linked by ln308 to generate an absolute module file.

### Format of relocatable module file

The relocatable module file generated by as308 is based on the IEEE-695 format.

- Since this file comes in a binary format, it cannot be output to a display screen or printer; nor can it be edited. Note that if you open or edit this file with an editor, file processing in the subsequent stages will not be performed normally.

### File name of relocatable module file

The file name of the relocatable module file is created by changing the extension of the assembly source file (.a308 by default) to "r30." (sample.a30 --> sample.r30)

### Directory for relocatable module file generated

If you specified the directory with command option (- O), the relocatable module file is generated in that directory. If no directory is specified, the relocatable module file is generated in the directory where the assembly source file resides.

## Assembler List File

Only when you specified command option (-L or -LM), as308 generates source line information and relocatable information as a file in text format that can be output to a display screen or printer .

### Format of assembler list file

The information listed below is output to an assembler list file. The output format of this assembler list file is shown in Example of Assembler List File -1.

- (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 : OXMSDA  
Outputs information on the results of source line processing performed by as308. Specifically, this information contains the following:

<i>0 XMSDA Contents</i>	
0-9	Indicates the include file's nest level.
x	Indicates that this line was not assembled in condition assemble.
M	Indicates that this is a macro 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.

### **File name of assembler list file**

The file name of the assembler list file is created by changing the extension of the assembly source file (.a30 by default) to ".lst" (sample.a30 --> sample.lst)

### **Directory for assembler list file generated**

If you specified the directory with command option (- O), the assembler list file is generated in that directory. If no directory is specified, the assembler list file is generated in the directory where the assembly source file resides.

## **Assembler Error Tag File**

Only when you specified command options (-T and -X), as308 outputs to a file the errors that were encountered when assembling the assembly source file.

### **Format of assembler error tag file**

The assembler error tag file is output in a format that allows you to use an editor's tag jump function.

This file is output in order of the assembly source file name, error line number, and error message as shown below.

```
sample.err 21 Error (asp308): Operand value is not defined
sample.err 72 Error (asp308): Undefined symbol exist "work2"
```

### **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)

### **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.

## Absolute Module File

The In308 editor generates one absolute module file from multiple relocatable module files.

### Format of absolute module file

This file is output in the format based on IEEE-695.

- Since this file comes in a binary format, it cannot be output to a screen or printer; nor can it be edited. Note that if you open or edit this file with an editor, file processing in the subsequent stages will not be performed normally.

### File name of absolute module file

The file name of the absolute module file normally is created by changing the extension ".r30" of the relocatable module file that is entered first from the command line into ".x30". (sample.r30 --> sample.x30)

If you specify a file name using command option (-O), the file is generated in specified name.

### Directory for absolute module file generated

The absolute module file normally is generated in the current directory.

If you specify a path in the file name of command option (-O), the absolute module file is generated in the directory of that path.

## Map File

Only when you specify command option (-M, -MS or -MSL), In308 outputs link information on last allocated section address, and symbol information to a map file. Symbol information is output only when you specify command option (-MS or -MSL).

### Format of map file

The information below is output to a map file sequentially in a list form. The output format of a typical map file is shown in Example of Map File.

- (1) Link information  
This information includes command lines, relocatable module file names, and the dates when the relocatable module files were created.
- (2) Section information  
This information includes the relocated section names, attributes, types, store addresses, section sizes, whether or not sections are aligned, and module names (relocatable module file names).
- (3) Global label information  
This information includes global label names and addresses. This information is output only when you specify command option "-MS/-MSL".
- (4) Global symbol information  
This information includes global symbol names and numeric values. This information is output only when you specify command option "-MS/-MSL".
- (5) Global bit symbol information  
This information includes global bit symbol names, bit positions, and memory addresses. This information is output only when you specify command option "-MS/-MSL".
- (6) Local label information  
This information includes module names (relocatable module file names), local label names, and addresses. This information is output only when you specify command option "-MS/-MSL".
- (7) Local symbol information  
This information includes module names (relocatable module file names), local symbol names, and numeric values. This information is output only when you specify command option "-MS/-MSL".
- (8) Local bit symbol information  
This information includes module names (relocatable module file names), local bit symbol names, bit positions, and memory addresses. This information is output only when you specify command option "-MS/-MSL".

**File name of map file**

The file name of the map file is created by changing the extension ".x30" of the absolute module file into ".map". (sample.x30 --> sample.map)

**Directory for map file generated**

The map file is generated in the directory where the absolute module file resides.

**Link Error Tag File**

Only when you specify command option (-T), ln308 outputs link error information to a file. In this case, locations in error are output with the assembly source lines.

**Format of link error tag file**

This file is output in the same format as an assembler error tag file. An editor's tag jump function can be used.

The link error tag file is output in order of the assembly source file name, error line number, and error message as shown below.

```
smp.inc 2 Warning (ln308): smp2.r30 : Absolute-section is written
after the absolute-section 'ppp'
smp.inc 2 Error (ln308): smp2.r30 : Address is overlapped in 'CODE'
section 'ppp'
```

**File name of link error tag file**

The file name of the link error tag file is created by changing the extension ".x30" of the absolute module file into ".ltg". (sample.x30 --> sample.ltg)

**Directory for link error tag file generated**

The link error tag file is generated in the directory where the absolute module file resides.

## Motorola S Format

The lmc308 generates a Motorola S format file that can be programmed into EPROM.

### Format of Motorola S file

The following can be specified when generating a Motorola S format file.

- Set the length of one data record to 16 bytes or 32 bytes.
- Set the start address of a program.

### File name of Motorola S file

The file name of the Motorola S file is created by changing the extension ".x30" of the absolute module file into ".mot". (sample.x30 --> sample.mot)

### Directory for Motorola S file generated

The files are generated in the current directory.

## Intel HEX Format

Only when you specify command option (-H), lmc308 generates an Intel HEX format file that can be programmed into EPROM.

### Format of Intel HEX file

The following can be specified when generating an Intel HEX format file.

- Set the length of one data record to 16 bytes or 32 bytes.
- IF the address value exceeds 1Mbytes of machine language file, the file of Original HEX format for Mitsubishi microcomputers is generated. This file can not be program into EPROM.

**File name of Intel HEX file**

The file name of the Intel HEX file is created by changing the extension ".x30" of the absolute module file into ".hex". (sample.x30 --> sample.hex)

**Directory for Motorola S file generated**

The files are generated in the current directory.

**Library File**

The lb308 librarian generates one library file from the relocatable module files generated by as308 by integrating them as modules into a single file.

**Format of library file**

The library file is based on the IEEE-695 format.

- Since this file comes in a binary format, it cannot be output to a screen or printer; nor can it be edited. If you open or edit this file with an editor, file processing in the subsequent stages will not be performed normally.

**File name of library file**

The library file is generated using the file name specified on the command line. The extension is ".lib". A library file name cannot be omitted on the command line.

**Directory for library file generated**

If a path is specified on the command line, the library file is generated in that directory. If no path is specified, the library file is generated in the current directory.



## Library List File

The lb308 librarian generates a list file indicating library files and the relocatable modules entered in each library file.

### Format of library list file

This file is output in a text format that can be output to a screen and printer. By referencing this file, it is possible to get approximate information about the relocatable modules entered in the library file. The format of a typical library list file is shown in Example of Library List File.

The following shows the information output to a library list file.

#### (1) Library file information

This information is output one for each library file. The library file information contains the following:

- **Library file name** (Library file name:)  
Indicates the library file name.
- **File update date and time** (Last update time:)  
Indicates the date and time the library file was updated last.
- **Number of modules** (Number of module(s):)  
Indicates the total number of modules entered in the library file.
- **Number of global symbols** (Number of global symbol(s): )  
Indicates the total number of global labels and global symbols entered in the library file.

## (2) Module information

This information is output one for each module entered in the library file. The module information contains the following:

- **Module name** (Module name:)  
Indicates the module names entered in the library file.
- **Version information** (.Ver: )  
Indicates a character string that is specified by the directive command ".VER".
- **Entered date and time** (Date:)  
Indicates the date and time when each module is entered in the library file.
- **Module size** (Size:)  
Indicates the code and data sizes of the modules entered in the library file.

■ These sizes differ from the file sizes of the relocatable module files.

- **Global symbol name** (Global symbol(s):)  
Indicates the global symbol and global label names defined in the modules.
- **External reference symbol name**  
Indicates the global symbol and global label names externally referenced by the module.

### Example of Library List file

```

Librarian (lb308) for M16C/80 Series Version 1.00.00
Library file name:      libsmp.lib
Last update time:      1995-Jul-7 15:44
Number of module(s):   1
Number of global symbol(s): 12

Module name:           sample
.Ver:                  .VER          "sample program file"
Date:                  1995-Jul-7 15:43
Size:                  00894H
Global symbol(s):      btsym5 btsym6 btsym7
                       btsym8 btsym9 sub1
                       sub2 sym5 sym6
                       sym7 sym8 sym9

```

## Cross Reference File

The xrf308 generates from the assembly source file a file that contains summary information on lines where symbols and labels are defined and referenced.

### Format of cross reference file

This file is output in a text format that can be output to a screen and printer. Therefore, you can print this file to a printer during debugging and check positions in the assembly source file where symbols are defined. The format of a typical cross reference file is shown in Example of Cross Reference File.

### Information in cross reference file

The following explains the information that is output to a cross reference file.

- (1) Label name  
This indicates a label name.
- (2) File name  
This indicates a file name in which the above label is written.
- (3) Reference line number and classification symbol  
This indicates a line number in which the label is defined and declared and a symbol denoting its classification as follows:
  - :d Definition line
  - :j Reference line for branch instruction
  - :s Reference line for subroutine call instruction

### Example of Cross Reference File

```
btsym0
  sample.a30
    00023:d
btsym1
  sample.a30
    00024:d
btsym2
  sample.a30
    00025:d
btsym20
  sample.a30
    00033:d
```

### **File name of cross reference file**

The file name of the cross reference file is created by changing the extension of the assembler list file (.lst) or assembly source file (.a30) to ".xrf". (sample.lst --> sample.xrf; sample.a30 --> sample.xrf)

However, if multiple file names are specified, the cross reference file name is derived from the first specified file name by changing its extension to ".xrf."

### **Directory for cross reference file generated**

If a path is specified on the command line, the cross reference file is generated in that directory.

If a directory is specified with command option (-O), the cross reference file is generated in that directory.

If a directory is not specified in neither way, the file is generated in the current directory.

## **Absolute List File**

The absolute list files generated by abs308 are output in a format that can be output to a screen or printer.

### **Format of absolute list file**

The absolute list file is the same format as that of the assembler list file except that location information is converted into absolute address information.

### **File name of absolute list file**

The file name of the absolute list file is derived by changing the extension of the assembler list file (.lst) to ".als". (sample.lst --> sample.als)

### **Directory for absolute list file generated**

If command option (-O) is specified, the absolute list file is generated in that directory.

Otherwise, the file is generated in the current directory.

# Starting Up Program

This section explains the basic method for operating each program included with AS308.

To operate any program included with AS308, always input a command from the prompt of your personal computer or workstation.

## Precautions on Entering Commands

- When using Windows, be sure to use the MS-DOS prompt to input a command.
- Although personal computers do not discriminate between uppercase and lowercase letters you input from the prompt, workstations are case sensitive. Therefore, when starting up each AS308 program on a workstation, always be sure to input program names using lowercase letters.
- Workstations discriminate between uppercase and lowercase letters in file names as they process files.

## Structure of Command Line

Input the following information on a command line.

### Program Name

This is the name of a program you want to use.

- When operating on a workstation, always be sure to input a program name using lowercase letters.

## Command Parameter

All information necessary to execute a program correctly is called "command parameters." For example, command parameters include the file names to be processed by the program you are going to start up and the command options that indicate program functions using symbols.

Command parameters include the following information:

- File name  
This means the name of a file to be processed by the program started up.
- When operating on a workstation, use uppercase and lowercase letters correctly as you input a file name.
- Command option  
Specify command options on the command line to use the functions of AS308 programs.

## Rules for Entering Command Line

When starting up each AS308 program, observe the rules for entering a command line described below.

### Number of Characters on Command Line

- The number of characters that can be entered on a command line is 128 characters (bytes) for the PC version or 512 characters (bytes) for the workstation version.
- The number of characters may be limited below the above specification depending on the operating environment (type of OS) of AS308.

### Method for Entering Command Line

- Always be sure to enter space between the startup program name and the file name.
- Always be sure to enter space between the file name and each command option.

### File Name

- The maximum length of a file name is 128 characters (bytes) including directory specification for the PC version or 512 characters (bytes) including directory specification for the workstation version. However, the number of characters on a command line must not exceed the above-mentioned size including the startup program name and all command options.
  - Descriptions of file names are subject to the naming conventions of the PC and workstation OS in addition to the above rules. Refer to the user's manual of your OS for details.
- Although workstations permit you to use a file name that is separated by the period (.) in two or more places, this does not apply when using AS308. AS308 allows use of the period in only one place of a file name. Furthermore, some AS308 programs restrict file name extensions (characters following the period) also. Refer to the method for starting up each AS308 program for details.

### Command Options

- Command options are not case sensitive regardless of whether you are operating on a PC or workstation. Therefore, they can be entered in either uppercase or lowercase.
- Always be sure to add a hyphen (-) when entering a command option.

# Method for Operating as308

This section explains the method for operating as308 to utilize its functions. The basic function of as308 is to generate a relocatable module file from the assembly source file.

## Command Parameters

The table below lists the command parameters of as308.

<i>Parameter name</i>	<i>Function</i>
Source file name	Source file name to be processed by as308.
-.	Disables message output to a display screen.
-abs16	To specify the 16-bit absolute addressing mode
-C	Indicates contents of command lines when as308 has invoked macro processor and assemble processor.
-D	Sets constants to symbols.
-F	Fixes the file name of ..FILE expansion to the source file name.
-H	Header information is not output to an assembler list file.
-L	Generates an assembler list file.
-mode60	To specify AS308 programs
-mode60p	To process structured commands for AS30
-M	Generates structured description command variables in byte type.
-N	Disables output of macro command line information.
-O	Specifies a directory to which the generated file is output.
-S	Specifies that localsymbol information be output.
-T	Generates an assembler error tag file.
-V	Indicates the version of the assembler system program.
-X	Invokes an external program as a tag file argument.



## Rules for Specifying Command Parameters

Follow the rules described below to specify the command parameters of as308.

### Order in which to specify command parameter

- Command parameters can be specified in any desired order.  
`as308 (assembly source file) (command option)`

### Assembly source file name (essential)

- Always be sure to specify one or more assembly source file names.
- A path can be specified for the assembly source file name.
- Up to 80 assembly source file names can be specified.
- If any of the multiple assembly source files thus specified contains an error, that file is not processed in the subsequent processing stages.
- Assembly source files with extension ".a30" can have their extensions omitted when you specify them.

### Command options

- Command options can be omitted.
- Multiple command options can be specified.
- Some command options allow you to specify a character string or a numeric value.
- Do not enter a space or tab between the command option and the character string or numeric value.
- If you want a subsequent command option to be nullified, add two consecutive hyphens (--) when entering that command option.
- Command options can only be nullified in as308. Therefore, this function cannot be used when starting up any other program.

Example:

- Option L only is valid.

```
as308 sample -L
```

- Option S only is valid.

```
as308 sample -S
```

- Option S only are valid.

```
as308 sample -L -S --L
```

- Options L only are valid.

```
as308 sample -S -L --S
```

### Method for specifying numeric value

- Always be sure to use hexadecimal notation when entering a numeric value.
- If a numeric value begins with an alphabet, always be sure to add 0 to the numeric value when you enter it.

Example:

```
55  
5A  
0A5
```

## **Include File Search Directory**

Include files do not need to be specified from the command line. If a path is described in the operand of the directive command ".INCLUDE", the software searches that directory to find the include file.

If the directive command operand does not have a path specification, the software searches the current directory. In this case, if the specified file cannot be found in the current directory and environment variable "INC308" is set, the software also searches the directory that is set in INC308.

## **as308 Command Options**

The following pages describe rules you need to follow when specifying command options.

-.

## Disables Message Output to Screen

### Function

- The software does not output messages when as308 is processing.
- This command option disables unnecessary messages such as copyright notes from being output to the screen when executing as308 in batch processing.
- Error messages, warning messages, and messages deriving from the directive command ".ASSERT" are output, however.

### Description rule

- This command option can be specified at any position on the command line.

### Description example

```
>as308 -. sample
```

If processing of sample resulted in generating an error, the following output will be obtained.

```
>as308 -. sample
```

```
sample.a30 2 Error (as308) : Section type is not appropriate
```

## -abs16

To specify the 16-bit absolute addressing mode

### Function

- When the below conditions are satisfied, AS308 selects 16-bit absolute addressing.
  - When the operand is a label or external reference symbol
  - When addressing mode is not specified
  - When the absolute addressing mode is the selected mnemonic statement
- Unless this option is specified, 24-bit absolute addressing is selected.

### Description rule

- Always input this option in small letters.
- The option can be specified at any point in the command line.

### Description example

```
>as308 -abs16 sample
```

## -C

Indicates Command Invocation Line

### Function

- In cases when a command option is specified in environment variable (AS308COM), if this option is specified you can confirm the command options set when invoking macroprocessor and assembler processor from as308 as the software indicates them on the screen.

### Description rule

- This option can be specified at any position on the command line.

### Description example

- If '-L -T' is set in AS308COM, the following output will be obtained.  

```
>as308 -C -N sample
```
- This information is displayed beginning with the next line following "All Rights Reserved." that is output when AS308 starts up normally.  

```
>as308 -C -N sample

( sample.a30 )
mac308 -L -T sample.a30
macro processing now

asp308 -L -T sample.a30
assembler processing now
TOTAL ERROR(S)      00000
:
```
- If this command option is combined with an option to disable message output to a screen, the following output will be obtained.  

```
>as308 -. -C sample
mac308 -L -T sample.a30

asp308 -L -T sample.m30
```

## -D

### Sets Symbol Constant

#### Function

- The software sets values to symbols.
- The value is handled as an absolute value.
- The symbols defined by this option are processed in the same way as those symbols that are defined in the start positions of the source program. However, these symbols are not output to an assembler list file.
- The symbols defined by this option are handled in the same way as the symbol definitions described in the assembly source file. Namely, if a symbol definition of the same name is described in the assembly source file, it means that the symbol is redefined at that description position.
- If multiple files are specified on the command line, the symbols defined by this option are handled as being defined in all of these files.

#### Description rule

- Specify this option in the form of -D (symbol name) = (numeric value).
- This option can be specified at any position on the command line.
- Do not enter a space or tab between the command option and the symbol name.
- Values can be defined to multiple symbols. When defining values to multiple symbols, separate each symbol with the colon while you enter them in a form like -D (symbol name) = (value): (symbol name) = (value): and so on.
- No space or tab can be entered in front or after the colon.

### Description example

- This example sets 1 to symbol name.  
`>as308 -Dname=1 sample`
- This example sets 1 to symbols name and symbol.  
`>as308 -Dname=1:symbol=1 sample`
- This example defines a symbol named name for files sample1 and sample2.  
`>as308 -Dname=1 sample1 sample2`



## -F

### Controls ..FILE Expansion

#### Function

- This option fixes the file name to be expanded by the directive command ..FILE to the assembly source file name that is specified from the command line.

#### Description rule

- This option can be specified at any position on the command line.

#### Description example

```
>as308 -F sample
```

The file name to be expanded by the directive command "..FILE" described in the "include.inc" file that is included by the sample.a308 assembly source file is fixed to "sample". If this option is not specified, the file name to be expanded by "..FILE" becomes "include".

## -H

Disable header output to an assembler list file

### Function

- Header information is not output to an assembler list file.
- When generating an assembler list file to be processed by as308, do not specify this option.

### Description rule

- This option can be written at any desired position in a command line.
- Specify this option simultaneously with the command option '-L.'

### Description example

- Header information is not output to the sample.lst file.
- ```
>as308 -L -H sample
```

**-I**

Specify an include file search directory

### Function

- The include file specified by ".INCLUDE" that is written in the source file is searched from a specified directory.

### Description rules

- This option can be written at any desired position in a command line.
- Specify a directory path immediately after "-I."
- No space or tab can be inserted between this option and a directory path name.

### Description example

- The include file written in the operand of a directive command ".INCLUDE" is searched from the \work\include directory.

```
>as308 -I\work\include
```

## -L

### Generates Assembler List File

#### Function

- The software generates an assembler list file in addition to a relocatable module file.
- The generated list files are identified by the extension ".lst".
- If a directory is specified by command option -O, the assembler list file is generated in the specified directory.

#### Description rule

- This option can be specified at any position on the command line.
- This option allows you to specify the 'I', 'M' and 'S' file format specifiers.
- No space or tab can be entered between the file format specifier and -L.
- Multiple file format specifiers can be specified simultaneously.
- File format specifiers can be entered in any desired order.
- This option can be set in environment variable "AS308COM".

| <i>Format specifier</i> | <i>Function</i>                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------|
| C                       | Line concatenation is output directly as is to a list file.                                                           |
| D                       | Information before .DEFINE is replaced is output to a list file.                                                      |
| I                       | Even program sections in which condition assemble resulted in false conditions are output to the assembler list file. |
| M                       | Even macro description expansion sections are output to the assembler list file.                                      |
| S                       | Even structured description for AS30 expansion sections are output to the assembler list file.                        |

#### Description example

```
>as308 -LIM sample
>as308 -CDLSMI sample
```

## -mode60

To specify AS308 programs

### Function

- Always specify this option to assemble programs written for AS30 (assembler for the M16C/60 Series) with AS308.
  - When this option is specified, AS308 replaces some of the commands in the program (written for AS30).
- For details, see "Compatibility with M16C/60 commands" and "AS308 processing when option command -mode60 is specified".

### Description rule

- Always input this option in small letters.
- The option can be specified at any point in the command line.

### Description example

- The program written for AS30 is reassembled as as308.  
`>as308 -mod60 sample`

## -mode60p

To process structured commands for AS30

### Function

- Always specify this option to structured command and assemble programs written for AS30 (assembler for the M16C/60 Series) with AS308.
- When this option is specified, AS308 converts structured command into assemble program for AS30 and then replace some of the commands in the program (written for AS30). For details, see "Compatibility with M16C/60 commands" and "AS308 processing when option command -mode60 is specified".

### Description rule

- Always input this option in small letters.
- The option can be specified at any point in the command line.

### Description example

- The structured commands in the assembly source file are processed and the developed part is output to an assembler list file.

```
>as308 -mod60p -LS sample
```

## -M

### Generate Structured Description Command Variables in Byte Type

#### Function

- The software processes variables in structured description commands whose types are indeterminate as the byte type.

#### Description rule

- This option can be specified at any position of the command line.
- Make sure this option is specified along with a command option "-P."

#### Description example

```
>as30 -P -M sample  
>as30 -M -P sample
```

## -N

Disables Line Information Output

### Function

- The software does not output C language source line information to a relocatable module file.
- The size of the relocatable module file can be reduced.
- Absolute module files generated from the relocatable module file that was generated after specifying this option cannot be debugged at the source line level.

### Description rule

- This option can be specified at any position on the command line.
- This option can be set in environment variable "AS308COM". Refer to "Example for using AS308COM" for details on how to set.

### Description example

```
as308 -N sample
```



## -O

Specifies Generated File Output Directory

### Function

- This option specifies the directory to which the relocatable module file, assembler list file, and assembler error tag file that are generated by the assembler are output.
- The directory name can be specified including a drive name. It can also be specified by a relative path.

### Description rule

- Write this option in the form of -O (directory name).
- No space or tab can be entered between this option and the directory name.

### Description example

- The relocatable module file is generated in the \work\asmout directory on drive c.  

```
>as308 -Oc:\work\asmout sample
```
- The relocatable module file is generated in the tmp directory that is the parent directory of the current directory.  

```
>as308 sample -O..\tmp
```
- The relocatable module file, assembler error tag file, and assembler list file are generated in the \work\asmout directory on drive c.  

```
>as308 -Oc:\work\asmout sample -L -T
```

## -S

### Specifies Local Symbol Information Output

#### Function

- The software outputs local symbol information to a relocatable module file.
- System label information can also be output a relocatable module file by adding 'M' to this option.
- Absolute module files generated from the relocatable module file that was generated after specifying this option can be symbolic debugged even for local symbols.

- The map file (.map) output by In308 provides information on symbolic debuggable symbols and labels so you can confirm.

#### Description rule

- If you want system label information and local label information to be output simultaneously, be sure to input this option as "-SM".
- This option can be specified at any position on the command line.
- This option can be set in environment variable "AS308COM". Refer to "Example for using AS308COM" for details on how to set.

#### Description example

- Local symbol information in sample.a30 is output to sample.r30.  
`>as308 -S sample`
- Local symbol information and system label information in sample.a30 is output to sample.r30.  
`>as308 -SM sample`

## -T

### Generates Assembler Error Tag File

#### Function

- The software generates an assembler error tag file when an assembler error is found.
- The file is output in a format where you can use an editor's tag jump function.
- Even when you have specified this option, no file will be generated if there is no error.
- The software does not generate a relocatable module file if an error is encountered. However, it does generate a relocatable module file in cases when only a warning has occurred.
- The error tag file name is created from the assembly source file name by changing its extension to ".atg".

#### Description rule

- This option can be specified at any position on the command line.
- This option can be set in environment variable "AS308COM". Refer to "Example for using AS308COM" for details on how to set.

#### Description example

- The software generates a "sample.atg" file if an error occurs.

```
>as308 -T sample
```

**-V**

Indicates Version Number

### Function

- When this option is specified, the software indicates the version numbers of all programs included with AS308 and terminates processing.
- All other parameters on the command line are ignored when this option is specified.

### Description rule

- Specify this option only and nothing else.

### Description example

```
>as308 -V
```

## -X

Invokes External Program

### Function

- After generating an assembler error tag file, the software invokes an execution program specified following the option '-X'.
- If this option is specified, the software generates an assembler error tag file when an error occurs regardless of whether or not you specified the option '-T'.

### Description rule

- Input this option using a form like -X (program name).
- No space or tab can be entered between this option and the program name.
- This option can be specified at any position on the command line.

### Description example

- The 'edit' is name of editor program.  

```
>as308 -Xedit sample
```

## Error Messages of as308

'#' is missing

- ? '#' is not entered.
- ! Write an immediate value in this operand.

')' is missing

- ? ')' is not entered.
- ! Write the right parenthesis ')' corresponding to the '('.

',' is missing

- ? ',' is not entered.
- ! Insert a comma to separate between operands.

'.B' or '.W' is not specified

- ? Neither .B nor .W is specified.
- ! Neither .B nor .W can be omitted. Write .B or .W in mnemonic.

'.IF' is missing for '.ELIF'

- ? .IF for .ELIF is not found.
- ! Check the position where .ELIF is written.

'.IF' is missing for '.ELSE'

- ? .IF for .ELSE is not found.
- ! Check the position where .ELSE is written.

'.IF' is missing for '.ENDIF'

- ? .IF for .ENDIF is not found.
- ! Check the position where .ENDIF is written.

' .MACRO' is missing for ' .ENDM'

- ? .MACRO for .ENDM is not found.
- ! Check the position where .ENDM is written.

' .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.

' .MACRO' or ' .MREPEAT' is missing for ' .EXITM'

- ? .MACRO or .MREPEAT for .EXITM is not found.
- ! Check the position where .EXITM is written.

' .MREPEAT' is missing for ' .ENDR'

- ? .MREPEAT for .ENDR is not found.
- ! Check the position where .ENDR is written.

' .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.

'ALIGN' is multiple specified in ' .SECTION'

- ? Two or more ALIGN's are specified in the .SECTION definition line.
- ! Delete extra ALIGN specifications.

'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.

'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.

'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.

'DEFAULT' has already defined

- ? There are multiple instances of DEFAULT in SWITCH.
- ! Remove unnecessary DEFAULT statements.

'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.

']' is missing

- ? ']' is not entered.
- ! Write the right bracket ']' corresponding to the '['.

Addressing mode specifier is not appropriate

- ? The addressing mode specifier is written incorrectly.
- ! Make sure that the addressing mode is written correctly.

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.

Can't create Temporary file

- ? Temporary file cannot be generated.
- ! Specify a directory in environment variable 'TMP308' so that a temporary file will be created in some place other than the current directory.

Can't create file 'filename'

- ? The 'filename' file cannot be generated.
- ! Check the directory capacity.



Can't open '.ASSERT' message file 'xxxx'

- ? The .ASSERT output file cannot be opened.
- ! Check the file name.

Can't open file 'filename'

- ? The 'filename' file cannot be opened.
- ! Check the file name.

Can't open include file 'xxxx'

- ? The include file cannot be opened.
- ! Check the include file name. Check the directory where the include file is stored.

Can't read file 'filename'

- ? The 'filename' file cannot be read.
- ! Check the permission of the file.

Can't write '.ASSERT' message file 'xxxx'

- ? Data cannot be written to the .ASSERT output file.
- ! Check the permission of the file.

Can't write in file 'filename'

- ? Data cannot be written to the 'filename' file.
- ! Check the permission of the file.

CASE not inside SWITCH

- ? CASE is written outside a SWITCH statement.
- ! Make sure the CASE statement is written within a SWITCH statement.

Characters exist in expression

- ? Extra characters are written in an instruction or expression.
- ! Check the rules to be followed when writing an expression.

Command line is too long

- ? The command line has too many characters.
- ! Re-input the command.

DEFAULT not inside SWITCH

- ? DEFAULT is written outside a SWITCH statement.
- ! Make sure the DEFAULT statement is written within a SWITCH statement.

Division by zero

- ? A divide by 0 operation is attempted.
- ! Rewrite the expression correctly.

ELSE not associates with IF

- ? No corresponding IF is found for ELSE.
- ! Check the source description.

ELIF not associates with IF

- ? No corresponding IF is found for ELIF.
- ! Check the source description.

ENDIF not associates with IF

- ? No corresponding IF is found for ENDIF.
- ! Check the source description.

ENDS not associates with SWITCH

- ? No corresponding SWITCH is found for ENDS.
- ! Check the source description.

Error occurred in executing 'xxx'

- ? An error occurred when executing xxx.
- ! Rerun xxx.

Format specifier is not appropriate

- ? The format specifier is written incorrectly.
- ! Make sure that the format specifier is written correctly.

Illegal directive command is used

- ? An illegal instruction is entered.
- ! Rewrite the instruction correctly.

Illegal file name

- ? The file name is illegal.
- ! Specify a file name that conforms to file name description rules.

Illegal macro parameter

- ? The macro parameter contains some incorrect description.
- ! Check the written contents of the macro parameter.

Illegal operand is used

- ? The operand is incorrect.
- ! Check the syntax for this operand and rewrite it correctly.

Include nesting over

- ? Include is nested too many levels.
- ! Rewrite include so that it is nested within the valid levels.

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.

Invalid bit-symbol exist

- ? An invalid bit symbol is entered.
- ! Rewrite the bit symbol definition.

## Invalid label definition

- ? An invalid label is entered.
- ! Rewrite the label definition.

## Invalid operand(s) exist in instruction

- ? The instruction contains an invalid operand.
- ! Check the syntax for this instruction and rewrite it correctly.

## Invalid option 'xx' is in environment data

- ? The environment variable contains invalid command option xx.
- ! Set the environment variable correctly back again. The options that can be set in environment variables are L, N, S, and T.

## Invalid reserved word exist in operand

- ? The operand contains a reserved word.
- ! Reserved words cannot be written in an operand. Rewrite the operand correctly.

## Invalid symbol definition

- ? An invalid symbol is entered.
- ! Rewrite the symbol definition.

## Invalid option 'xx' is used

- ? An invalid command option xx is used.
- ! The specified option is nonexistent. Re-input the command correctly.

## Location counter exceed 0FFFFFFH

- ? The location counter exceeded 0FFFFFFh.
- ! Check the operand value of .ORG. Rewrite the source correctly.

## NEXT not associates with FOR

- ? No corresponding FOR is found for NEXT.
- ! Check the source description.

No 'ENDIF' statement

- ? No corresponding ENDIF is found for the IF statement in the source file.
- ! Check the source description.

No 'ENDS' statement

- ? No corresponding ENDS is found for the SWITCH statement in the source file.
- ! Check the source description.

No 'NEXT' statement

- ? No corresponding NEXT is found for the FOR statement in the source file.
- ! Check the source description.

No 'WHILE' statement

- ? No corresponding WHILE is found for the DO statement in the source file.
- ! Check the source description.

No '.END' statement

- ? .END is not entered.
- ! Be sure to enter .END in the last line of the source program.

No '.ENDIF' statement

- ? .ENDIF is not entered.
- ! Check the position where .ENDIF is written. Write .ENDIF as necessary.

No '.ENDM' statement

- ? .ENDM is not entered.
- ! Check the position where .ENDM is written. Write .ENDM as necessary.

No '.ENDR' statement

- ? .ENDR is not entered.
- ! Check the position where .ENDR is written. Write .ENDR as necessary.

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.

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.

No '.SECTION' statement

- ? .SECTION is not entered.
- ! Always make sure that the source program contains at least one .SECTION.

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.

No input files specified

- ? No input file is specified.
- ! Specify an input file.

No macro name

- ? No macro name is entered.
- ! Write a macro name for each macro definition.

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.

Not enough memory

- ? Memory is insufficient.
- ! Divide the file and re-run. Or increase the memory capacity.

Operand expression is not completed

- ? The operand description is not complete.
- ! Check the syntax for this operand and rewrite it correctly.

Operand number is not enough

- ? The number of operands is insufficient.
- ! Check the syntax for these operands and rewrite them correctly.

Operand size is not appropriate

- ? The operand size is incorrect.
- ! Check the syntax for this operand and rewrite it correctly.

Operand type is not appropriate

- ? The operand type is incorrect.
- ! Check the syntax for this operand and rewrite it correctly.

Operand value is not defined

- ? An undefined operand value is entered.
- ! Write a valid value for operands.

Option 'xx' is not appropriate

- ? Command option xx is written incorrectly.
- ! Specify the command option correctly again.

## Questionable syntax

- ? The structured description command is written incorrectly.
- ! Check the syntax and write the command correctly again.

## Quote is missing

- ? Quotes for a character string are not entered.
- ! Enclose a character string with quotes as you write it.

## Reserved word is missing

- ? No reserved word is entered.
- ! Write a reserved word [SB], [FB], [A1], [A0], [SP], or [A1A0].

## Reserved word is used as label or symbol

- ? Reserved word is used as a label or symbol.
- ! Rewrite the label or symbol name correctly.

## Right quote is missing

- ? A right quote is not entered.
- ! Enter the right quote.

## Same items are multiple specified

- ? Multiple same items of operand are specified.
- ! Check the syntax for this operand and rewrite it correctly.

## 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.

## 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.



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.

Section name is missing

- ? No section name is entered.
- ! Write a section name in the operand.

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.

Section type is not appropriate

- ? The section type is written incorrectly.
- ! Rewrite the section type correctly.

Size or format specifier is not appropriate

- ? The size specifier or format specifier is written incorrectly.
- ! Rewrite the size specifier or format specifier correctly.

Size specifier is missing

- ? No size specifier is entered.
- ! Write a size specifier.

Source files number exceed 80

- ? The number of source files exceeds 80.
- ! Execute assembling separately in two or more operations.

Source line is too long

- ? The source line is excessively long.
- ! Check the contents written in the source line and correct it as necessary.

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.

String value exist in expression

- ? A character string is entered in the expression.
- ! Rewrite the expression correctly.

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.

Symbol definition is not appropriate

- ? The symbol is defined incorrectly.
- ! Check the method for defining this symbol and rewrite it correctly.

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.

Symbol has 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.

Symbol is missing

- ? Symbol is not entered.
- ! Write a symbol name.

Symbol is multiple defined

- ? The symbol is defined twice or more. The macro name and some other name are duplicates.
- ! Change the name.

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.

Syntax error in expression

- ? The expression is written incorrectly.
- ! Check the syntax for this expression and rewrite it correctly.

Temporary label is undefined

- ? The temporary label is not defined yet.
- ! Define the temporary label.

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.

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.

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.

## 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.

## 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.

## Too many operand

- ? There are extra operands.
- ! Check the syntax for these operands and rewrite them correctly.

## 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.

## Too many temporary label

- ? There are too many temporary labels.
- ! Replace the temporary labels with label names.

## Undefined symbol exist

- ? An undefined symbol is used.
- ! Define the symbol.

## Value is out of range

- ? The value is out of range.
- ! Write a value that matches the register bit length.

## WHILE not associates with DO

- ? No corresponding DO is found for WHILE.
- ! Check the source description.

## Warning Messages of as308

`'.ALIGN' with not 'ALIGN' specified relocatable section`

- ? Directive command ".ALIGN" is written in a section that does not have an ALIGN specification.
- ! 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.

`'CASE' definition is after 'DEFAULT'`

- ? CASE is preceded by a DEFAULT description.
- ! Make sure all DEFAULT commands are written after the CASE statement.

`'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.

`'.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.

`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.

Addressing is described by the numerical value

- ? Addressing is specified with a numeric value.
- ! Be sure to write '#' in numeric values.

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.

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 address.

Fixed data in 'CODE' section

- ? Found directive command(.BYTE, .WORD, .ADDR, .LWORD) in the section type is CODE.
- ! Specify ROMDATA type the section written any directive command(.BYTE, .WORD, .ADDR, .LWORD).

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.

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.

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.

Mnemonic in 'ROMDATA' section

- ? Found mnemonic in the section type is ROMDATA.
- ! Specify CODE type to the section written mnemonic.

Moved between address registers as byte size

- ? Transfers between address registers are performed in bytes.
- ! Rewrite the mnemonic correctly.

Statement has not effect

- ? The statement does not have any effect as a command line.
- ! Check the correct method for writing the command.

Too many actual macro parameters

- ? There are too many actual macro parameters.
- ! Extra macro parameters will be ignored.

Too many structured label definition

- ? There are too many labels to be generated.
- ! Divide the file into smaller files before assembling.

Unnecessary BREAK is found

- ? Found two or over BREAK statement in a SWITCH block.
- ! Check the source program.

# Method for Operating In308

This section describes how to use the functions of In308. The basic function of In308 is to generate one absolute module file from two or more relocatable module files.

## Command Parameters

The table below lists the command parameters available for In308.

| <i>Parameter name</i> | <i>Function</i>                                                             |
|-----------------------|-----------------------------------------------------------------------------|
| File name             | Relocatable module filename to be processed by In308                        |
| -.                    | Disable message output to screen.                                           |
| -E                    | Specifies start address of absolute module.                                 |
| -G                    | Outputs source debug information to absolute module file.                   |
| -L                    | Specifies library file to be referenced.                                    |
| -LOC                  | Specifies section allocation sequence.                                      |
| -LD                   | Specifies directory of library to be referenced.                            |
| -M                    | Generates map file.                                                         |
| -MS                   | Generates mapfile that includes symbol information.                         |
| -MSL                  | Generates mapfile that includes fullname of symbol more than 16 characters. |
| -NOSTOP               | Outputs all encountered errors to screen.                                   |
| -O                    | Specifies absolute module file name.                                        |
| -ORDER                | Specifies section address and allocation sequence.                          |
| -T                    | Generates link error tag file.                                              |
| -V                    | Indicates version number of linkage editor.                                 |
| @                     | Specifies command file.                                                     |



## Rules for Specifying Command Parameters

Follow the rules described below when you specify command parameters for In308.

### Order in which to specify command parameters

- Relocatable module file names and command options can be specified in any desired order.

```
>ln308 (command options) (relocatable module file)
>ln308 (relocatable module file) (command options)
```

### Relocatable module file name (essential)

- Always be sure to specify at least one relocatable module file name.
- A path can be specified in the file name.
- When specifying multiple relocatable module files, always be sure to insert a space or tab between each file name.

### Absolute module file name

- Normally In308 creates the file name of an absolute module file from the relocatable module file that is specified first as it generates the absolute module file.
- Use command option (-O) to specify an absolute module file name.

### Library file name

- Use command option (-L) to specify the library file to be referenced. A path can be specified in the file name.
- Library files are searched from the directory that is set in environment variable (LIB308). If the relevant file cannot be found, In308 searches the current directory. Or if a directory is specified by command option (-LD), In308 searches it and if no relevant file is found in this directory, In308 searches the current directory.

### Command option

- When you specify a command option, always be sure to insert a space or tab between the command option and other specifications on the command line.

### Address specification

- The In308 editor determines absolute addresses section by section as it generates an absolute module file.
- When invoking In308, you can specify the start address of a section from the command line.
- Use hexadecimal notation when specifying address values. If an address value begins with an alphabet, add 0 to the value as you specify it.

Example:

```
7fff
64
0a57
```

## Command File

The In308 editor allows you to write command parameters in a file and execute the program after reading in this file.

### Method for specifying command file name

- Add @ at the beginning of the command file name as you specify it.

Example:

```
>ln308 @cmdfile
```

- A directory path can be specified in the command file name.
- If no file exists in the specified directory path, In308 outputs an error.

## Rules for writing command file

The following explains the rules you need to follow when writing a command file to ensure that it can be processed by In308.

- The name of the command file's own cannot be written in the command file.
- Multiple lines of command parameters can be written in a command file.
- The comma (,) cannot be entered at the beginning and end of lines written in a command file.
- If you want to write specification for section allocation in multiple lines, be sure to enter the "-ORDER" option at the beginning of each new line.
- The maximum number of characters that can be written on one line in the file is 255 characters. If this limit is exceeded, In308 outputs an error.
- Comments can be written in a command file. When writing comments, be sure to enter the symbol "#" at the beginning of each comment. Characters from # to Carriage Return or Line Feed are handled as comments.

## Example of command file description

```
sample1 sample2
sample3
-ORDER ram=80
-ORDER prog, sub, datasub, and data
-M
```

-.

Disables Message Output to Screen

### Function

- The software does not output messages when In308 is processing.
- Error messages are output to screen.

### Description rule

- This option can be specified at any position on the command line.

### Description example

```
ln308 -. sample1 sample2
```

## -E

Specifies Start Address of Program

### Function

- This option sets the entry address of an absolute object module. This address is used to indicate the start address to the debugger.
- Numeric values or label names can be used to specify an address value. However, local label names cannot be specified.

### Description rule

- Input this option using a form like -E (numeric value or label name).
- Always be sure to insert a space between this option and the numeric value or label name.
- Always be sure to use hexadecimal notation when entering a numeric value.
- If the numeric value begins with an alphabet ('a' to 'f'), always be sure to add 0 at the beginning of the value as you enter it.
- This option can be specified at any position on the command line.

### Description example

- The address value in global label "num" is specified for the entry address of "sample1.x30".

```
ln308 sample1 sample2 -E num
```

- f0000 is specified for the entry address of "sample1.x30".

```
ln308 sample1 sample2 -E 0f0000
```

## -G

### Outputs Source Debug Information

#### Function

- The software outputs information on C language or macro description source lines to an absolute module file.
- The absolute module files generated without specifying this option cannot be debugged at the source line level.
- If the absolute module file is derived by linking the relocatable module files that were generated by specifying option (-N) to disable line information output when executing as308, it cannot be debugged at the source line level even when you have specified this option (-G) when executing ln308.
- Source debug information is output to an absolute module file.

#### Description rule

- This option can be specified at any position on the command line.

#### Description example

```
ln308 -G sample1 sample2
```

## -L

Specifies Library File Name

### Function

- Specify the library file name to be referenced when linking files.
- The In308 editor reads global symbol information from the specified library file as it links the necessary relocatable modules.

### Description rule

- Input this option using a form like -L (library file name).
- Always be sure to insert a space between this option and the file name.
- This option can be specified at any position on the command line.
- A path can be specified in the file name.
- Multiple library files can be specified. When specifying multiple library files, separate each file name with the comma as you specify file names. There must be no space or tab before or after the comma.

### Description example

```
>ln308 sample1 sample2 -L lib1
```

The "lib1.lib" file in the current directory or the directory specified in environment variable (LIB308) is referenced as necessary.

```
>ln308 sample1 sample2 -L work\lib1
```

The "lib1.lib" file in the "work" directory that resides below the current directory.

```
>ln308 sample1 sample2 -L lib1,lib2
```

The "lib1.lib" and "lib2.lib" files in the current directory or the directory specified in environment variable (LIB308) are referenced as necessary.

## -LD

Specifies Library File Directory

### Function

- Specify the directory name in which you want a library file to be referenced.
- Even when you specify this option, you need to specify the library file name.
- The directory name specified by this option remains valid until another directory is specified by this option next time.
- If you have specified a path in the library file name, the directory in which library files are referenced by ln308 is one that is located by linking the library file path to the directory specified by this option.

### Description rule

- Input this option using a form like -LD (directory name).
- Always be sure to insert a space between this option and the directory name.
- This option can be specified at any position on the command line.

### Description example

- The `\work\lib\lib1` file is referenced.  

```
>ln308 sample1 sample2 -LD \work\lib -L lib1
```
- The `\work\lib\lib1` and `\work\tmp\lib2` files are referenced.  

```
>ln308 sample1 sample2 -LD \work\lib -L lib1 -LD \work\tmp -L lib2
```
- The `\work\lib\lib1` file is referenced.  

```
>ln308 sample1 -LD \work -L lib\lib1
```



## -LOC

Specify the assignment of section

### Function

- Specifies the address in which the specified section is written.
- Value of symbols in specified section are generated from the address specified by directive command ".ORG" or specified by command option "-ORDER".
- Use this option if you write the section on address other than executing it.

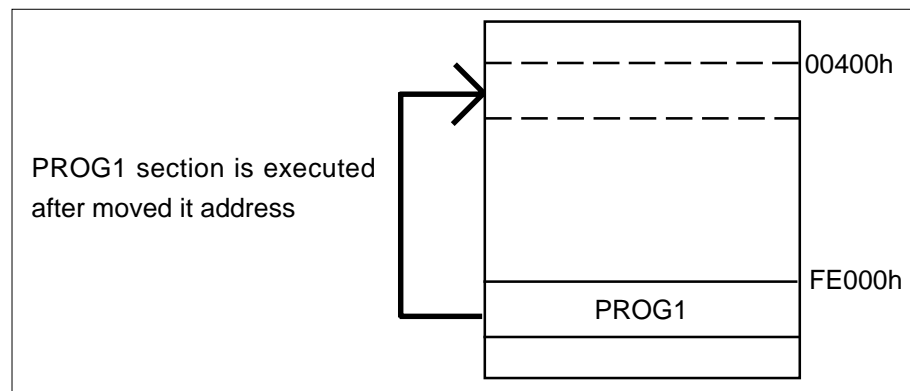
### Description rule

- A space is required between the option name and parameter.
- No space is allowed before and after the "=".
- The address cannot be omitted.
- When writing multiple section names and location addresses, separate each entry with a comma (,).

### Description example

- This example describes writing PROG1 section executed from 400h address on 0FE00h address of ROM.

```
>ln308 -ORDER PROG1=00400 -LOC PROG1=0FE000
```



## -M

Generates Map File

### Function

- The software generates a map file that contains address mapping information.
- The file name of the map file is created by changing the extension of the absolute module file to ".map".

### Description rule

- This option can be specified at any position on the command line.

### Description example

- Files "sample1.x30" and "sample1.map" are generated.  

```
>ln308 -M sample1 sample2
```

## -MS/-MSL

Outputs Symbol Information to Map File

### Function

- The fullname of symbol more than 16 characters are output to mapfile when -MSL is specified.
- The software generates a map file that contains address mapping information and symbol information.
- The file name of the map file is created by changing the extension of the absolute module file to ".map".

### Description rule

- This option can be specified at any position on the command line.

### Description example

```
>ln308 sample1 sample2 -MS
```

```
    A "sample1.x30" and "sample1.map" files are  
generated.
```

## **-NOSTOP**

Outputs All Errors

### **Function**

- The software outputs all encountered link errors to the screen.
- If this operation is not specified, the software outputs up to 20 errors to the screen.

### **Description rule**

- This option can be specified at any position on the command line.

### **Description example**

```
>ln308 sample1 sample2 -NOSTOP
```

## -O

Specifies Absolute Module File Name

### Function

- This option allows you to specify any desired name for the absolute module file generated by ln308.
- If you do not specify an absolute module file name using this option, the file name of absolute module file is created by changing to ".x30" the extension of the relocatable module file name that is specified first on the command line.

### Description rule

- Input this option using a form like -O (file name).
- Always be sure to insert a space between this option and the file name.
- The extension of a file name can be omitted. If omitted, the extension is ".x30".
- A path can be specified in the file name.

### Description example

- A "abssmp.x30" file is generated.  

```
>ln308 sample1 sample2 -O abssmp
```
- A "abssmp.x30" file is generated in the "\work\absfile" directory.  

```
>ln308 -O \work\absfile\abssmp sample1 sample2
```

## -ORDER

Specifies Section Address and Relocation Order

### Function

- Specify the order in which you want sections to be allocated and the start address of sections.
- If the start address is specified for an absolute section, ln308 outputs an error.
- If you do not specify the start address, ln308 allocates addresses beginning from 0.
- If sections of the same name exist in the specified relocatable files, sections are allocated in the order the files are specified. In this case, if a section with absolute attribute is arranged after a section with relative attribute, an error results.

### Description rule

- Input this option using a form like -ORDER (section name), (section name) or -ORDER (section name) = (start address).
- Always be sure to insert a space between this option and the section name.
- Separate between two section names or between an address value and a section name with a comma as you specify them. There must be no space or tab before or after the comma.
- This option can be specified at any position on the command line.

### Description example

- Sections are allocated in order of main, sub, and dat beginning from address 0H.  

```
>ln308 sample1 sample2 -ORDER main,sub,dat
```
- Sections are allocated in order of main, sub, and dat beginning from address 0f000H.  

```
>ln308 sample1 sample2 -ORDER main=0f0000,sub,dat
```

## -T

### Generates Link Error Tag File

#### Function

- The software generates a link error tag file when a link error occurs.
- This file is output in a format that allows you to use an editor's tag jump function.
- Even when you specify this option, this error file will not be generated if no error is encountered.
- The error tag file name is created from the relocatable module file that is specified at the beginning of the command line by changing its extension to ".ltg". If an absolute module file name is specified with command option "-O," the tag file name is derived from the specified file name by changing its extension to ".ltg."
- Error information in the link error tag file is output with the number of assembly source lines.

#### Description rule

- This option can be specified at any position on the command line.

#### Description example

- A "sample1.ltg" file is generated if an error occurs.  

```
>ln308 sample1 sample2 -T
```

**-V**

Indicates Version Number

### Function

- The software indicates the version number of ln308.
- All other parameters on the command line are ignored when this option is specified.

### Description rule

- Specify this option only and nothing else.

### Description example

```
>ln308 -V
```



@

Specifies Command File

### Function

- The software starts up In308 by using the contents of the specified file as the command parameters.

### Description rule

- Input this option using a form like @ (file name).
- No space or tab can be entered between this option and the file name.
- No other parameters can be written on the command line.

### Description example

```
>ln308 @cmdfile
```

## Error Messages of In308

'-loc' section 'section' is multiple defined

- ? The section name specified by the -loc option here has already been defined before .
- ! Check the section name.

'-loc' section 'section' is not found

- ? The section specified by the -loc option cannot be found.
- ! Check the section name.

'-order' section 'section' is multiple defined

- ? The section name specified with -order is defined twice or more.
- ! Make sure that sections are defined only once.

'-order' section 'section' is not found

- ? The section specified with -order cannot be found.
- ! Check the section name and re-run.

'CODE' section 'section-1' is overlapped on the 'section-2'

- ? The CODE sections 'section-1' and 'section-2' are overlapping.
- ! Relocate the sections so that they will not overlap.

'ROMDATA' section 'section-1' is overlapped on the 'section-2'

- ? The ROMDATA sections 'section-1' and 'section-2' are overlapping.
- ! Relocate the sections so that they will not overlap.

'section' is written after the same name of relocatable section

- ? A relative attribute section is followed by an absolute attribute section of the same name 'section'.
- ! Make sure that relative attribute is located after absolute attribute.

'symbol' is multiple defined

- ? The symbol 'symbol' is defined twice or more.
- ! Check external symbol names.

'symbol' value is undefined

- ? The value of the symbol 'symbol' is not defined yet.
- ! The program will be processed assuming values = 0. Check the symbol values.

Absolute section 'section' is relocated

- ? Absolute section 'section' is going to be relocated.
- ! Correct the section locating specification.

Address is overlapped in 'CODE' section 'section'

- ? Addresses are overlapping in a CODE section named 'section'.
- ! Relocate the section so that its addresses will not overlap.

Address is overlapped in 'ROMDATA' section 'section'

- ? Addresses are overlapping in a ROMDATA section named 'section'.
- ! Relocate the section so that its addresses will not overlap.

Can't close file 'file'

- ? The file 'file' cannot be closed.
- ! Check the directory information.

Can't close temporary file

- ? The temporary file cannot be closed.
- ! Check the remaining storage capacity of the disk.

Can't create file 'file'

- ? The file 'file' cannot be created.
- ! Check the directory information.

Can't create temporary file

- ? A temporary file cannot be created.
- ! Check to see if the directory is write protected.

Can't open file 'file'

- ? The file 'file' cannot be opened.
- ! Check the file name.

Can't open temporary file

- ? The temporary file cannot be opened.
- ! Check the directory information.

Can't remove file 'file'

- ? The file 'file' cannot be deleted.
- ! Check the permission of the file.

Can't remove temporary file

- ? The temporary file cannot be deleted.
- ! Check the permission of the file.

Can't registered symbol in the list

- ? Symbols cannot be registered in a list.
- ! If this error occurs, please contact tool support personnel at Mitsubishi.

Command-file line characters exceed 255

- ? The number of characters per line in the command file exceeds 255.
- ! Check the contents of the command file.

Command line is too long

- ? The command line contains too many characters.
- ! Create a command file.

DEBUG information mismatch in file

- ? Some file whose format version of relocatable module file does not match that of other file is included.
- ! Redo assembling using the latest assembler.

Illegal file extension '.xxx' is used

- ? The file extension '.xxx' is illegal.
- ! Specify a correct file extension.

Illegal format 'file'

- ? The format of the file 'file' is illegal.
- ! Check to see that the relocatable file is one that was created by as308.

Illegal format 'file' :expression error occurred

- ? The format of the file 'file' is illegal.
- ! Check to see that the relocatable file is one that was created by as308.

Illegal format 'file' :it's not library file

- ? The format of the file 'file' is illegal. That is not a library file.
- ! Check to see that the library file is one that was created by lb308.

Illegal format 'file' :it's not relocatable file

- ? The format of the file 'file' is illegal. That is not a relocatable file.
- ! Check to see that the relocatable file is one that was created by as308.

Invalid option 'option' is used

- ? An invalid option 'option' is used.
- ! Specify a correct option.

MCU information mismatch in file 'file'

- ? The MCU information in the file 'file' does not match the actual chip.
- ! Check to see that the relocatable file is one that was created by as308.

MCU information mismatch in file xx.r30

- ? A file generated by as30 is being used.
- ! Reassemble the file with as308 before linking.

No input files specified

- ? No input file is specified.
- ! Specify a file name.

Not enough memory

- ? Memory capacity is insufficient.
- ! Increase the memory capacity.

Option 'option' is not appropriate

- ? The option 'option' is used incorrectly.
- ! Check the syntax for this option and rewrite it correctly.

Option parameter address exceed 0FFFFFFH

- ? The address specified with an option exceeds 0FFFFFFh.
- ! Re-input the command correctly.

symbol type of floating point is not supported

- ? Floating-point representation of the symbol type is not supported.
- ! If this error occurs, please contact tool support personnel at Mitsubishi.

Zero division exists in the expression

- ? Expression for relocation data calculations contain a divide by 0 operation.
- ! Rewrite the expression correctly.

## Warning Messages of In308

'-e' option parameter 'symbol' is undefined

- ? The symbol 'symbol' specified with -e is not defined yet.
- ! Define 'symbol' in the source program. The program will be processed assuming values = 0.

'CODE' section 'section-1' is overlapped on the 'section-2'

- ? The CODE section 'section-1' overlaps 'section-2.' The sections have been allocated overlapping each other.
- ! Check to see if these sections are allowed to overlap.

'DATA' section 'section-1' is overlapped on the 'section-2'

- ? The DATA sections 'section-1' and 'section-2' are overlapping. Sections are located overlapping each other.
- ! Check to see if the sections can be located at overlapping addresses.

'ROMDATA' section 'section-1' is overlapped on the 'section-2'

- ? The ROMDATA section 'section-1' overlaps 'section-2.' The sections have been allocated overlapping each other.
- ! Check to see if these sections are allowed to overlap.

'label' value exceed 0FFFFFFH

- ? The value of the label 'label' exceeds 0FFFFFFh.
- ! Check the allocated addresses of sections.

'section' data exceed 0FFFFFFH

- ? The section data exceeds address 0FFFFFFH.
- ! Check the allocated addresses of sections.

16-bits signed value is out of range -32768 -- 32767 address  
='address'

? Relocation data calculation resulted in the address exceeding the range  
of -32,768 to +32,767.

! Overflow is discarded. Check whether the value is all right.

16-bits unsigned value is out of range 0 -- 65535 address='address'

? Relocation data calculation resulted in the address exceeding the range  
of 0 to 65,535.

! Overflow is discarded. Check whether the value is all right.

16-bits value is out of range -32768 -- 65535 address='address'

? Relocation data calculation resulted in the address exceeding the range  
of -32,768 to +65,535.

! Overflow is discarded. Check whether the value is all right.

24-bits signed value is out of range -8388608 --8388607  
address='address'

? Relocation data calculation resulted in the address exceeding the range  
of -8,388,608 to +8,388,607.

! Overflow is discarded. Check whether the value is all right.

24-bits unsigned value is out of range 0 -- 16777215  
address='address'

? Relocation data calculation resulted in the address exceeding the range  
of 0 to 16,777,215.

! Overflow is discarded. Check whether the value is all right.

24-bits value is out of range -8388608 -- 16777215  
address='address'

? Relocation data calculation resulted in the address exceeding the range  
of -8,388,608 to 16,777,215.

! Overflow is discarded. Check whether the value is all right.



- 4-bits signed value is out of range -8 -- 7 address='address'
- ? Relocation data calculation resulted in the address exceeding the range of -8 to 7.
  - ! Overflow is discarded. Check whether the value is all right.
- 8-bits signed value is out of range -128 -- 127 address='address'
- ? Relocation data calculation resulted in the address exceeding the range of -128 to 127.
  - ! Overflow is discarded. Check whether the value is all right.
- 8-bits unsigned value is out of range 0 -- 255 address='address'
- ? Relocation data calculation resulted in the address exceeding the range of 0 to 255.
  - ! Overflow is discarded. Check whether the value is all right.
- 8-bits value is out of range -128 -- 255 address='address'
- ? Relocation data calculation resulted in the address exceeding the range of -128 to 255.
  - ! Overflow is discarded. Check whether the value is all right.
- Absolute-section is written after the absolute-section 'section'
- ? The absolute attribute section 'section' is followed by an absolute attribute of the same name. The source program may be allocated at noncontinued addresses.
  - ! Linkage will be executed. Check the address specification of the source program.
- Absolute-section is written before the absolute-section 'section'
- ? The absolute attribute is concatenated before the absolute attribute section 'section'.
  - ! Concatenation will be executed. Check address specification in the source program.

Address information mismatch in file 'file'

- ? The address information in the relocatable file 'file' does not match the addresses information.
- ! Check to see that the relocatable file is one that was generated by as308.

Address is overlapped in the same 'DATA' section 'section'

- ? Addresses are overlapping in the DATA sections of the same name 'section'. The sections are located overlapping one another.
- ! Check to see if the sections can be located at overlapping addresses.

JMP.S instruction exist at end of bank(address xxxxx)

- ? The jump address of a short-jump instruction overlaps a bank boundary.
- ! Use the directive command '.SJMP' to control code generation so that short-jump instructions will not be generated at such a position.

Object format version mismatch in file 'file'

- ? The version information in the relocatable file or library file 'file' does not match the version information.
- ! Check to see that the relocatable file or library file is one that was generated by the AS308 program. Regenerate the file as necessary. If this error occurs, please contact tool support personnel at Mitsubishi.

Section type mismatch 'section'

- ? Sections of the same name 'section' have different section types.
- ! Check the section types in the source file.

# Method for Operating Imc308

This section explains how to operate Imc308. The primary function of Imc308 is to generate a machine language file in the Motorola S format from the absolute module files generated by In308.

## Command Parameters

The table below lists the command parameters available for Imc308.

| <i>Parameter name</i> | <i>Function</i>                                      |
|-----------------------|------------------------------------------------------|
| File name             | Absolute module file name to be processed by Imc308. |
| -.                    | Disables message output to screen.                   |
| -E                    | Sets the starting address.                           |
| -H                    | Converts file into Intel HEX format.                 |
| -ID                   | Set ID code for ID check function                    |
| -L                    | Selects maximum length of data record area.          |
| -O                    | Specifies output file name.                          |
| -V                    | Indicates version of load module converter.          |
| -protect1             | Set level1 for ROM code protect function             |
| -protect2             | Set level2 for ROM code protect function             |

## Rules for Specifying Command Parameters

Follow the rules described below when you specify the command parameters for lmc308.

### Order in which to specify command parameters

Always be sure to specify command parameters in the following order:

- 1 Command option
- 2 Absolute module file name (essential)

```
>lmc308 (command option) (absolute module file name)
```

### Absolute module file name (essential)

- Specify the absolute module file generated by ln308.
- Specify only one absolute module file name.
- The file extension (.x30) can be omitted.
- No file names can be specified unless their extension is ".x30".

### Command options

- Specify command options as necessary.
- Multiple command options can be specified.
- When specifying multiple command options, the command options can be entered in any order.

-.

Disables Message Output to Screen

### Function

- The software does not output messages when lmc308 is processing.
- Error messages are output to screen.

### Description rule

- Always be sure to specify this option before the file name.

### Description example

```
>lmc308 -.. debug
```

## -E

Sets the Starting Address

### Function

- Set the starting address.
- Output to a Motorola S format file beginning with the address you have set.
- The Motorola S format file is output with the setting starting address.

### Description rule

- Input this option using a form like -E (address value).
- Always be sure to insert a space between this option and the value.
- Always be sure to use hexadecimal notation when specifying an address value.
- If the address value begins with an alphabet ('a' to 'f'), always be sure to add 0 at the beginning of the value as you enter it.

■ This option cannot be specified simultaneously with "- H".

### Description example

```
>lmc308 -E 0f0000 debug
```

A "debug.mot" file is generated that starting address is 0f000H.

```
>lmc308 -E 8000 debug
```

A "debug.mot" file is generated that starting address is 8000H.

## -H

Converts File into Intel HEX Format

### Function

- The lmc308 generates an Intel HEX format file.
- The lmc308 generates an Original HEX format for Mitsubishi microcomputers if the address value exceeds 1Mbytes.

### Description rule

- Specify this option before entering a file name.
- This option cannot be specified simultaneously with option "-E".

### Description example

```
>lmc308 -H debug
```

## -ID

Set ID code for ID check function

### Function

- For details on the ID code check, see the hardware manual of the micro-computer.
- The specified ID code is stored as 8-bit data in ID store addresses (FFFFDF, FFFFE3, FFFFEB, FFFFEF, FFFFF3, FFFFF7 and FFFFFB). And FF is stored in address FFFFFFFF (Refer to -protect1).
- If you filled in ID store addresses with value in your source program, when this option is specified, the data of ID store addresses are always changed. Without this option, the filling data are output.
- When this option alone is specified, ID code is FFFFFFFFFFFFFFFF.
- An ID file (extension .id) is created to display ID codes set with this option.
- The specified ID code is stored as an ASCII code.
- Always specify this command option in capital letters.
- Add "-ID" to the ID code.
- To directly specify an ID code, specify "-ID#" followed by a number.

#### Example 1) -IDCodeNo1

ID code: 436F64654E6F31

| Address | FFFFDF | FFFE3 | FFFEB | FFFEF | FFFF3 | FFFF7 | FFFFB |
|---------|--------|-------|-------|-------|-------|-------|-------|
| data    | 43     | 6F    | 64    | 65    | 4E    | 6F    | 31    |

#### Example 2)-IDCode

ID code: 436F6465000000

#### Example 3)-ID1234567

ID code: 31323334353637

#### Example 4)-ID#49562137856132

ID code: 49562137856132

#### Example 5)-ID#1234567

ID code: 12345670000000

#### Example 6)-ID

ID code: FFFFFFFFFFFFFFFF



**-L**

Selects Maximum Length of Data Record Area

**Function**

- The data record length of the Motorola S format is set to 32 bytes.
- The data record length of the Intel HEX format is set to 32 bytes.

**Description rule**

- Specify this option before entering a file name.

**Description example**

```
>lmc308 -L debug
```

## -O

Specifies Output File Name

### Function

- Specify the file name of the machine language file generated by lmc308.
- A path can be specified in the file name.

### Description rule

- Input this option using a form like -O (file name).
- Always be sure to insert a space between this option and the file name.
- Specify this option before entering a file name.
- The file name cannot be specified with an extension. A default extension is used for the generated file: ".mot" for the Motorola S format and ".hex" for the Intel HEX format.

### Description example

```
>lmc308 -O test debug
```

A "test.mot" file is generated.

```
>lmc308 -O tmp\test debug
```

A "test.mot" file is generated in the "tmp" directory.

**-V**

Indicates Version Number

### Function

- The software indicates the version number of lmc308.
- If this option is specified, all other parameters on the command line are ignored.

### Description rule

- Specify this option only and nothing else.

### Description example

```
>lmc308 -V
```

## -protect1

Set level1 for ROM code protect function

### Function

- For details on the ROM code protect function, see the hardware manual of the microcomputer.
- 3F is stored in protect code store address FFFFFF.
- If you filled in protect code store address with value, when this option is specified, the protect code is changed. When this option is not specified, filling value is output.
- When options (-ID, -protect1, -protect2) to use ROM protect function is specified, the following protect code is filled in protect code store address.

| -ID     | -protect1 | -protect2 | Protect code                    |
|---------|-----------|-----------|---------------------------------|
| Specify | Non       | Non       | FF                              |
| Specify | Specify   | Non       | 3F                              |
| Specify | Non       | Specify   | F3                              |
| Specify | Specify   | Specify   | lmc308 error                    |
| Non     | Specify   | Non       | 3F                              |
| Non     | Non       | Specify   | F3                              |
| Non     | Non       | Non       | Value filling in source program |

### Description rule

- Always specify this command option in small letters.
- The protect2 option cannot be specified at the same time as the protect1 option.

### Description example

```
>lmc308 -protect1 sample
```

## -protect2

Set level2 for ROM code protect function

### Function

- For details on the ROM code protect function, see the hardware manual of the microcomputer.
- F3 is stored in protect code store address FFFFFF.
- If you filled in protect code store address with value, when this option is specified, the protect code is changed. When this option is not specified, filling value is output.
- When options (-ID, -protect1, -protect2) to use ROM protect function is specified, the following protect code is filled in protect code store address.

| -ID     | -protect1 | -protect2 | Protect code                    |
|---------|-----------|-----------|---------------------------------|
| Specify | Non       | Non       | FF                              |
| Specify | Specify   | Non       | 3F                              |
| Specify | Non       | Specify   | F3                              |
| Specify | Specify   | Specify   | lmc308 error                    |
| Non     | Specify   | Non       | 3F                              |
| Non     | Non       | Specify   | F3                              |
| Non     | Non       | Non       | Value filling in source program |

### Description rule

- Always specify this command option in small letters.
- The protect1 option cannot be specified at the same time as the protect2 option.

### Description example

```
>lmc308 -protect2 sample
```

## Error Messages of Imc308

'-e' option is too long

- ? The array of -e option parameters is excessively long.
- ! Check the syntax for this option and rewrite it correctly.

'xxx' option multiple specified

- ? The option 'xxx' is specified twice or more.
- ! Check the syntax for this option and rewrite it correctly.

Address specified by '-e' option exceed 0FFFFH

- ? The address specified with -e option exceeds 0FFFFh.
- ! Rewrite the address value correctly.

Can't close file 'filename'

- ? The file 'filename' cannot be closed.
- ! Check the directory information.

Can't create file 'filename'

- ? The file 'filename' cannot be created.
- ! Check the directory information.

Can't open file 'filename'

- ? The file 'filename' cannot be opened.
- ! Check the file name.

Command line is too long

- ? The character string on the command line is excessively long.
- ! Re-input the command correctly.

Illegal file format 'filename' is used

- ? The file format of 'filename' is incorrect.
- ! Check the file name. Regenerate the file.

Invalid option 'option' is used

- ? An invalid option 'option' is specified.
- ! Specify the option correctly again.

MCU information mismatch in file xx.x30

- ? The file information does not match the MCU information.
- ! Specify an absolute module file generated by as308 or ln308.

Not enough memory

- ? Memory is insufficient.
- ! Increase the memory capacity.

Option 'option' is not appropriate

- ? The option is used incorrectly.
- ! Check the syntax for this option and rewrite it correctly.

Unknown file extension '.xxx' is specified

- ? The specified file extension '.xxx' is incorrect.
- ! Check the file name.

## Warning Messages of Imc308

'filename' does not contain object data

- ? The specified file does not contain object data.
- ! Check the file name.

Address exceed 0FFFFFFH

- ? The address exceeded 0FFFFFFh.
- ! Check the written contents of the source program. Check to see how sections are located.



# Method for Operating lb308

This section explains the method for operating lb308 to utilize its functions. The primary function of lb308 is to manage multiple relocatable module files as a single library file.

## Command Parameters

The table below lists the command parameters available for lb308.

| <i>Parameter name</i> | <i>Function</i>                                        |
|-----------------------|--------------------------------------------------------|
| File name             | Relocatable module file name to be processed by lb308. |
| -.                    | Disable message output to screen.                      |
| -A                    | Adds module to library file.                           |
| -C                    | Creates new library file.                              |
| -D                    | Deletes modules from library file.                     |
| -L                    | Generates library list file.                           |
| -R                    | Replaces modules.                                      |
| -U                    | Updates modules.                                       |
| -V                    | Indicates version of librarian.                        |
| -X                    | Extracts modules.                                      |
| @                     | Specifies command file.                                |

## Rules for Specifying Command Parameters

Follow the rules described below when you specify command parameters for lb308.

### Order in which to specify command parameters

Always specify the command parameters for lb308 in the following order. If the command parameters are specified in an incorrect order, lb308 cannot process files correctly.

- 1 Command option
- 2 Library file name
- 3 Relocatable module (file) name

```
lb308 (command option) (library file name) (relocatable module file name)
```

### Library file name (essential)

- Always be sure to specify the library name.
- A directory path can be specified in the file name.
- The extension (lib) can be omitted on the command line.

### Relocatable module file name (relocatable module name)

- Always be sure to specify a relocatable module file name.
- The extension of a relocatable module file name is '.r30'. The extension can be omitted on the command line.
- Multiple relocatable module files can be specified. In this case, always be sure to insert a space between each file name.
- A directory path can be specified in the file name. If no directory is specified, the files residing in the current directory are processed.

**Command options**

- Command options are not case sensitive. They can be entered in uppercase or lowercase.
- At least one of the command options '-A', '-C', '-D', '-L', '-R', '-U', or '-X' must always be specified when executing the librarian. If none of these options is specified on the command line or two or more of them are specified simultaneously, lb308 outputs an error.

**Command File**

- The librarian allows you to specify a command file name that contains description of input parameters.
- Refer to the Method for Operating In308 for details on how to specify a command file.

-.

## Disables Message Output to Screen

### Function

- The software does not output messages when lb308 is processing.
- Error messages are output to screen.

### Description rule

- This option alone can be specified in combination with some other options.
- This option and other options can be specified in any order.

### Description example

```
lb308 -. -A new sample2
```

## -A

### Adds Modules to Library File

#### Function

- The software adds a relocatable module to an existing library file.
- If the specified library file is nonexistent, lb308 creates a new library file.
- If a relocatable module bearing the same name as one you are going to add is already entered in the library file, lb308 outputs an error.
- If the relocatable module file you are going to add contains a definition of the same global symbol name as in the module that is already entered in the library file, lb308 outputs an error.

#### Description rule

- Input this option using a form like -A (library file name) (relocatable module file name).
- Always be sure to insert a space between this option and the library file name and between the library file name and the relocatable module file name.

#### Description example

```
lb308 -A new.lib sample3.r30
```

A "sample3" module is added to the "new.lib" file.

## -C

### Creates New Library File

#### Function

- The software creates a new library file.
- If a library file of the same name as one you have specified in this command option already exists, the contents of the old library file are replaced with those of the new library file.

#### Description rule

- Input this option using a form like -C (library file name) (relocatable module file name).
- Always be sure to insert a space between this option and the library file name and between the library file name and the relocatable module file name.

#### Description example

```
lb308 -C new sample1 sample2
```

A new library file named "new.lib" is created that contains sample1 and sample2.

## -D

### Deletes Modules from Library File

#### Function

- The software deletes a specified relocatable module from the library file.
- Once deleted, the module is nonexistent anywhere.

#### Description rule

- Input this option using a form like -D (library name) (relocatable module name).
- Always be sure to insert a space between this option and the library file name and between the library file name and the relocatable module name.
- Multiple relocatable modules you want to be deleted can be specified. In this case, always be sure to insert a space between each module name.

#### Description example

```
lb308 -D new sample2
```

A relocatable module "sample2" is deleted from the "new.lib" library file.

## -L

### Generates Library List File

#### Function

- The software generates a library list file that contains information on a specified library file. The extension of generated library list file is ".lls".
- A library list file can also be generated that contains information on only the necessary modules in the library file.
- If a library list file of the same name already exists, this existing file is overwritten by a new library list file.

#### Description rule

- Input this option using a form like -L (library file name) [(relocatable module name)].
- Always be sure to insert a space between this option and the library file name and between the library file name and the relocatable module file name.
- Multiple relocatable module names can be specified. In this case, always be sure to insert a space between each module name.

#### Description example

```
lb308 -L new
```

Information on all modules entered in a library file named "new.lib" are output to a library list file named "new.lls".

```
lb308 -L new sample1
```

Information on module sample1 entered in the "new.lib" library file is output to a "new.lls" list file.

```
lb308 -L new.lib sample1 sample3
```

Information on modules sample1 and sample3 entered in the "new.lib" library file are output to a "new.lls" list file.



## -R

### Replaces Modules

#### Function

- The software updates a relocatable module in the library file by replacing it with the content of a specified relocatable module file. The module that is updated in this way is one that has the same name as the specified relocatable module file name.

#### Description rule

- Input this option using a form like -R (library file name) (relocatable module file name).
- Always be sure to insert a space between this option and the library file name and between the library file name and the relocatable module file name.
- Multiple relocatable module file names can be specified. In this case, always be sure to insert a space between each module file name.

#### Description example

```
lb308 -R new sample1
```

The content of module sample1 in the "new.lib" library file is replaced with the content of the "sample1.r30" file of the same name.

## -U

### Updates Modules

#### Function

- The software compares the created date of a relocatable module in the library file with that of a relocatable module file with which you want to be updated. Then if the date of the relocatable module file is newer than that of the module, the software updates it.

#### Description rule

- Input this option using a form like -U (library file name) (relocatable module file name).
- Always be sure to insert a space between this option and the library file name and between the library file name and the relocatable module file name.
- Multiple relocatable module names can be specified. In this case, always be sure to insert a space between each module name.

#### Description example

```
lb308 -U new sample1
```

Only when the created date of module sample1 in the "new.lib" file is older than that of the "sample1.r30" file of the same name, the content of sample1 is updated with the content of the "sample1.r30" file.

**-V**

Indicates Version Number

### Function

- The software outputs the version number of lb308 to the screen.
- If this option is specified, all other parameters on the command line are ignored.

### Description rule

- Specify this option only and nothing else.

### Description example

```
lb308 -V
```

## -X

### Extracts Module

#### Function

- The software extracts a relocatable module from the library file as a relocatable module file.
- The library file is not modified by this operation.
- The created date of the relocatable module file thus extracted is the date when it was extracted from the library file.
- If a file of the same name as the extracted relocatable module file already exists, the existing file is overwritten.

#### Description rule

- Always be sure to insert a space between this option and the library file name.

#### Description example

```
lb308 -X new sample3
```

Module sample3 is extracted from the "new.lib" library file to generate a relocatable module file named "sample3.r30".

@

Specifies Command File

### Function

- The software uses the contents of a specified file as command parameters as it invokes lb308.

### Description rule

- Input this option using a form like @ (file name).
- No space or tab can be entered between this option and the file name.
- No other parameters can be entered on the command line.

### Description example

```
lb308 @cmdfile
```

## Error Messages of Ib308

'filename' is not library file

- ? The file 'filename' is not a library file.
- ! Check the file name. Check to see that the file is one that was generated by Ib308.

'filename' is not relocatable file

- ? The file 'filename' is not a relocatable file.
- ! Check the file name. Check to see that the file is one that was generated by as308.

'module' already registered in 'filename'

- ? The module 'module' has already been registered in the library 'filename'.
- ! Check the library file name and the relocatable file name.

'module' does not match with 'filename'

- ? The module name 'module' and the relocatable file name 'filename' do not match. The module name has been modified.
- ! Check the relocatable file name.

'module' is multiple specified

- ? Multiple modules of the same name 'module' are specified.
- ! Specify the module name correctly again.

'module' is not registered in 'filename'

- ? The module 'module' is not registered in the library file 'filename'. Specified processing (to delete or update module) cannot be performed.
- ! Check the module name.

'symbol' is multiple defined at 'module1' and 'module2' in 'filename'

- ? Externally defined symbols of the same name 'symbol' are defined in two places of the library 'filename', one in 'module1' and another in 'module2'.
- ! Check the relocatable file name.

'symbol' is multiple defined in 'filename'

- ? The symbol 'symbol' is defined twice in the file 'filename'.
- ! If this error occurs, please contact tool support personnel at Mitsubishi.

'symbol' is multiple defined in 'module1' and 'module2'

- ? Externally defined symbol 'symbol' is defined in two places of the library 'filename', one in 'module1' and another in 'module2'.
- ! Check the relocatable file name.

'xxx' and 'xxx' are used

- ? The option 'xxx' and the option 'xxx' are used simultaneously.
- ! Options cannot be specified simultaneously. Re- input the command correctly.

Can't close file 'filename'

- ? The file 'filename' cannot be closed.
- ! Check the directory information.

Can't close temporary file

- ? The temporary file cannot be closed.
- ! Check the directory information.

Can't create file 'filename'

- ? The file 'filename' cannot be created.
- ! Check the directory name.

Can't create temporary file

- ? The temporary file cannot be created.
- ! Check the directory information.

Can't open file 'filename'

- ? The file 'filename' cannot be opened.
- ! Check the file name.

Can't open temporary file

- ? The temporary file cannot be opened.
- ! Check the directory information.

Can't write in file 'filename'

- ? Data cannot be written to the file 'filename'. Memory is insufficient.
- ! Increase the memory capacity.

Command-file is include in itself

- ? An attempt is made to include the command file in itself.
- ! Check to see if the command file is written correctly.

Command-file line characters exceed 255

- ? The number of characters per line in the command file exceeded 255 characters.
- ! Check the contents of the command file.

Command line is too long

- ? The character string on the command line is excessively long.
- ! Create a command file.

Illegal file format 'filename'

- ? The file format of 'filename' is incorrect.
- ! Check the file name.



Invalid option 'option' is used

- ? An invalid option 'option' is used.
- ! Specify the option correctly again.

MCU informatino mismatch in file xx.r30

- ? The file information does not match the MCU information.
- ! Specify a relocatable module file generated by as308.

No public symbol is in 'filename'

- ? There is no public symbol in the file 'filename'.
- ! Check the contents of the relocatable file.

Not enough memory

- ? Memory is insufficient.
- ! Increase the memory capacity.

Symbol-name characters exceed 500

- ? The symbol name consists of more than 500 characters.
- ! Divide the library file.

Too many modules

- ? There are too many registered modules.
- ! Divide the library file into two or more files.

Unknown file extension '.xxx' is used

- ? The file extension '.xxx' is incorrect.
- ! Check the file name.

## Warning Messages of Ib308

'module' is not registered in library

- ? The module 'module' is not registered in the library. Therefore, no modules of the specified name were extracted.
- ! Check the module name.

'module' is not registered in library, can't output list-file

- ? The module 'module' is not registered in the library. Information on this module was not output to a list file.
- ! Check the module name.

'module' was created in the current directory

- ? The module 'module' was created in the current directory.
- ! Check the directory name you have specified.

Can't replace, 'module' is older than module in library

- ? The module 'module' is older than the module in the library. Therefore, the library module was not replaced with it.
- ! Check the created date of the relocatable file.

# Method for Operating xrf308

This section explains the method for operating xrf308 to utilize its functions. The basic function of xrf308 is to generate from the assembly source file or assembler list file a cross reference file that contains a list for referencing branch instructions and subroutine call instructions.

## Command Parameters

The table below lists the command parameters available for xrf308.

| <i>Parameter name</i> | <i>Function</i>                                               |
|-----------------------|---------------------------------------------------------------|
| File name             | Source or assembler list file name to be processed by xrf308. |
| -.                    | Disables message output to screen.                            |
| -N                    | Specifies that system label information be output.            |
| -O                    | Specifies directory in which to output a file.                |
| -V                    | Indicates version of cross referencer.                        |
| @                     | Specifies command file.                                       |

## Rules for Specifying Command Parameters

Follow the rules described below when specifying the command parameters of xrf308.

### Order in which to specify command parameters

The command parameters of xrf308 can be specified in any order.

```
>xrf308 (file name) (command option)
>xrf308 (command option) (file name)
```

**Assembly source file name or assembler list file name**

- Always be sure to specify at least one file name.
- A path can be specified in the file name.
- Up to 600 files can be specified.
- Always be sure to enter the file extension.
- Always be sure to specify assembler list file whose extension is ".lst".
- When specifying multiple files, insert a space or tab to separate between file names.

**Command options**

- Multiple command options can be specified.

**Command File**

- The xrf308 referencer allows you to specify a command file name that contains input parameters.
- Refer to the Method for Operating In308 for details on how to specify a command file.

-.

Disables Message Output to Screen

### Function

- The software does not output messages when xrf308 is processing.
- Error messages are output to screen.

### Description rule

- This option can be specified at any position on the command line.

### Description example

```
xrf308 -. sample.a30
```

## -N

Specifies Output of System Label Information

### Function

- Information on system labels output by as30 also is output to a cross reference file.
- System labels are one that begins with two periods (..).

### Description rule

- This option can be specified at any position on the command line.

### Description example

```
xrf308 -N sample.lst
```

A "sample.xrf" file is generated from a "sample.lst" file.

```
xrf308 -N sample.a30
```

A "sample.xrf" file is generated from a "sample.a30" file

## -O

Specifies File Output Directory

### Function

- Specify a directory in which you want the cross reference file to be output.

### Description rule

- Input this option using a form like -O (directory name).
- No space or tab can be entered between this option and the directory name.
- This option can be specified at any position on the command line.

### Description example

```
xrf308 -O\work\list sample.a30
```

A "sample.xrf" file is generated in a \work\list directory.

```
xrf308 -O\work\list sample.lst
```

**-V**

Indicates Version Number

### Function

- The software indicates the version number of the cross referencer.
- If this option is specified, all other parameters on the command line are ignored.

### Description rule

- Specify this option only and nothing else.

### Description example

```
xrf308 -V
```



@

Specifies Command File

### Function

- The software uses the contents of a specified file as command parameters as it invokes xrf308.

### Description rule

- No space or tab can be entered between this option and the file name.
- No other parameters can be entered on the command line.

### Description example

```
xrf308 @cmdfile
```

## Error Messages of xrf308

Can't create temporary file

- ? The temporary file cannot be created.
- ! Check the directory information.

Can't open file 'xxxx'

- ? The 'xxxx' file cannot be opened.
- ! Check the file name.

Command-file is included in itself

- ? An attempt is made to include the command file in itself.
- ! Check the written contents of the command file.

Command-file line characters exceed 255

- ? The number of characters per line in the command file exceeds 255 characters.
- ! Check the contents of the command file.

Command line is too long

- ? The character string on the command line is excessively long.
- ! Create a command file.

Input files exceed 80

- ? The number of input files exceeds 80.
- ! Re-input the command. Divide the contents of the command file.

Invalid option 'xxx' is used

- ? An invalid option 'option' is specified.
- ! Specify the command option correctly again.

No input files specified

- ? No input file is specified.
- ! Specify a file name.

Not enough memory

- ? Memory is insufficient.
- ! Increase the memory capacity.

Option 'xxx' is not appropriate

- ? The command option is specified incorrectly.
- ! Check the syntax for this command option and specify it correctly again.

# Method for Operating abs308

## Precautions using abs308

- If two or more same section declarations exist and the section is not output to the assembler list file by the directive command ".LSIT OFF" in one assembly sourcefile, a correct actual address might not be generated.
- Specify command option "-LM" when the assembler as308 processed the source file that contains macro directive command.
- Specify command option "-LS" when the assembler as308 processed the source file that contains structured directive command for AS30.
- It is needed that header lines are output to assembler list file. Operate as308 without command option -H.

## Command Parameters

The table below lists the command parameters available for abs308.

| <i>Parameter name</i> | <i>Function</i>                                                       |
|-----------------------|-----------------------------------------------------------------------|
| File name             | Assembler list or absolute modulefile name to be processed by abs308. |
| -.                    | Disables message output to screen.                                    |
| -D                    | Specifies directory in which to search files.                         |
| -O                    | Specifies directory in which to output files.                         |
| -V                    | Indicates version of absolute lister.                                 |

## Rules for Specifying Command Parameters

Follow the rules described below when specifying command parameters.

### Order in which to specify command parameters

- Always be sure to specify command parameters in the order given below:
  - 1 Command option
  - 2 Absolute module file name
  - 3 Assembler list file name

```
>abs308 (command option) (absolute module file name) (assembler list file name)
```

### File name of absolute module file (essential)

- Always be sure to specify the absolute module file name.
- A path can be specified in the absolute module file name.
- The extension (.x30) can be omitted.

**File name of assembler list file**

- Multiple assembler list files can be specified by separating them with a space or tab.
- A path can be specified in the assembler list file name.
- The file attribute can be omitted.
- The assembler list file name can be omitted.

**Command options**

- Command options are not case sensitive, so they can be entered in uppercase or lowercase.
- Always be sure to enter a space or tab between the command option and its argument.

-.

Disables Message Output to Screen

### Function

- The software does not output messages when xrf308 is processing.
- Error messages are output to screen.

### Description rule

- This option can be specified at any position on the command line.

### Description example

```
xrf308 -. sample.a30
```

## -D

Specifies File Search Directory

### Function

- Specify the directory in which you want assembler list files to be searched.
- If this directory is not specified, abs308 searches assembler list files from the current directory.

### Description rule

- Input this option using a form like -D (directory name).
- No space or tab can be entered between this option and the directory name.

### Description example

```
abs308 sample -Ddir
```

Assembler list files in "dir" under the current directory are searched.

```
abs308 sample -Ddir list1
```

File "list1.lst" is searched in "dir" under the current directory is searched.



## -O

Specifies File Output Directory

### Function

- Specify the directory in which you want the absolute list file to be generated.
- If this directory is not specified, the absolute list file is generated in the current directory.

### Description rule

- Input this option using a form like -O (directory name).
- No space or tab can be entered between this option and the directory name.

### Description example

```
abs308 sample -Oabslist
```

The absolute list file is generated in the "abslist" directory under the current directory.

**-V**

Indicates Version Number

### Function

- The software indicates the version number of the absolute lister.
- If this option is specified, all other parameters on the command line are ignored.

### Description rule

- Specify this option only and nothing else.

### Description example

```
abs308 -V
```

## Error Messages of abs308

Can't create file 'filename'

- ? The file 'filename' cannot be created.
- ! Check the directory information.

Can't open file 'filename'

- ? The file 'filename' cannot be opened.
- ! Check the file name.

Can't write in file 'filename'

- ? Data cannot be written to the file 'filename'.
- ! Check the permission of the file.

Command line is too long

- ? The command line contain too many characters.
- ! Re-input the command correctly.

Error information is in 'filename'

- ? The file 'filename' contains error information.
- ! Regenerate the assembler list file.

Illegal file format 'filename'

- ? The file format of 'filename' is illegal.
- ! Check the file name.

Input files number exceed 80

- ? The number of input files exceeds 80.
- ! Re-input the command.

Not enough disk space

- ? Disk capacity is insufficient.
- ! Check the disk information.

Not enough memory

- ? Memory capacity is insufficient.
- ! Increase the memory capacity.

Section information is not appropriate in 'filename'

- ? The section information in 'filename' is incorrect.
- ! Check the file name.

## Warning Messages of abs308

Address area exceed 0FFFFFFH

- ? The address range exceeds 0FFFFFFh.
- ! Check the absolute module file name.

File 'l-filename' is missing corresponding to module in 'a-filename'

- ? The file 'l-filename' corresponding to the module in 'a-filename' cannot be found. The absolute list file for this module was not created.
- ! Regenerate the assembler list file. Check the directory where the assembler list file resides.

Lines 'num-num' are relocatable address in 'filename'

- ? The lines 'num-num' in 'filename' not converted to absolute addresses.
- ! Check to see if the directive command ".LIST OFF" is written in the assembly source file.

No information of 'l-filename' in 'a-filename'

- ? The file 'a-filename' does not contain information on 'l-filename'.
- ! Check the file name.

Overwrite in 'filename'

- ? The file 'filename' will be overwritten.
- ! The contents of the old file are not saved anywhere.

# Rules for Writing Program

This section describes the basic rules you need to follow when writing a source program that can be assembled by AS308.

## Precautions on Writing Program

Pay attention to the following when writing a program to be assembled by AS308:

- Do not use the reserved words of AS308 for names in your source program.
- The character strings consisting of AS308 directive commands which have had the periods removed can be used for names without causing an error. However, avoid using these character strings because some of them affect processing performed by AS308.
- System labels (the character strings that begin with "..") written in your source program may not result in generating an error. However, avoid using system labels because some of them may be used for AS308 extension in the future.

## Character Set

You can use the following characters when writing an assembly program to be assembled by AS30.

### 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

### Lowercase 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

### Numerals

0 1 2 3 4 5 6 7 8 9

### Special characters

" # \$ % & ' ( ) \* + , - . / : ; [ ] \ ^ \_ | ~

### Blank

(Space) (Tab)

### New paragraph or line

(Carriage retn) (Line feed)

- Always be sure to use 'en'-size characters when writing instructions and operands. You cannot use multi-byte characters (e.g., kanji) unless you are writing comments.

## Reserved Words

AS308 handles the same character strings as directive assemble commands and mnemonics as reserved words. These reserved words are not case-sensitive, so they are not discriminated between uppercase and lowercase. Consequently, "ABS" and "abs" are the same reserved words.

- The reserved words cannot be used in the "names" described later.

### Types of Reserved Words

- Directive assemble commands  
All directive assemble commands explained in this manual and all character strings that begin with one period are the reserved words.
- Mnemonic  
All assembly language mnemonic of M16C/80 Series are the reserved word.
- Operators  
All operators explained in this manual are the reserved word.
- System labels  
The system labels are generated by assembler.  
AS308 handles all character strings that begin with two period as system labels.



## Names

Any desired names can be defined and used as such in your source program. Names are classified into the following types, each with a different range of descriptions that can be entered in the program.

- Label  
This name has an address as its value.
- Symbol  
This name has a constant as its value.
- Bit symbol  
This name has a constant (bit position) and address as its values.
- Location symbol  
This name has an address as its value. This symbols are output by as308.

### Rules for Writing Names

#### Length of name

A character string can be entered as a name in up to 255 characters.

#### Determination of name

Names are case-sensitive, so they are discriminated between uppercase and lowercase. Therefore, "LAB" and "Lab" are handled as different names.

- You cannot use any name that is identical to one of AS308's reserved words. If this rule is not followed, program operation cannot be guaranteed.

The following describes the types of names you can define in your program.

## Label

### Function

- This is a name assigned to a specific address in the range of addresses that can be accessed by the CPU.

### Rules for writing

- Alphabets, numerals and the underline can be used for this name.
- Numerals cannot be used at the beginning of this name.
- When defining a name, always be sure to add the colon (:) at the end of the name.

### Defining method

- There are two methods to define a label.
  - 1 Allocate a memory area with a directive command.

Example:

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

- 2 Write a name at the beginning of a source line.

Example:

```
name1:  
_name:  
sym_name:
```

### Referencing method

Write a name in the operand of a mnemonic.

Example:

```
JMP sym_name
```

## Symbol

### Function

- This is a name assigned to a constant.

### Rules for writing

- Numeric values must be a determined value when assembling the source program.
- Alphabets, numerals and the underline can be used for this name.
- Numerals cannot be used at the beginning of this name.
- This name can be defined outside the range of sections.

### Defining method

- To define a symbol, use a directive command that is used for defining numeric values.

Example:

```
value1    .EQU 1
value2    .EQU 2
```

Referencing method

Write a symbol in the operand of an instruction.

Example:

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

## Bit symbol

### Function

- This is a name assigned to a specific bit position in specific memory.
- If this name is assigned to each individual bit in 8-bit long memory, one-byte memory can have 8 pieces of information.
- The bit position thus specified is offset from the least significant bit of memory specified in the address part by a value specified in the bit number part.

### Rules for writing

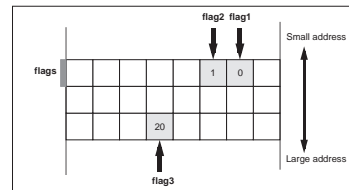
- Numeric values must be a determined value when assembling the source program.
- Alphabets and the underline can be used for this name.
- Numerals cannot be used at the beginning of this name.
- This name can be defined outside the range of sections.

### Defining method

- To define a bit symbol, use a directive command that is used for defining bit symbols.

Example:

```
flag1      .BTEQU      1,flags
flag2      .BTEQU      2,flags
flag3      .BTEQU     20,flags
```



Referencing method

A bit symbol can be written in the operand of a 1-bit operation instruction.

Example:

```
BCLR flag1
BCLR flag2
BCLR flag3
```

## Location symbol

### Function

- This symbol indicates the address of a line you wrote.
- By writing the dollar mark (\$) in the operand, you can indicate the address of the first byte of op-code in the line you wrote.

### Rules for writing

- Write this symbol in the operand of a mnemonic.
- The dollar mark (\$) cannot be written at the beginning of a name or reserved word.
- A location symbol can be written in a term of an expression.

- When writing a location symbol, make sure that the value of the expression is a valid value when your program is assembled.

### Description example

```
JMP .B $+5
```

- 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 address.

## Lines

The as308 assembler processes the source program one line at a time. Lines in the source program are classified into the following types depending on the contents in that line.

### Directive command line

- This line is where as308's directive command is written.
  - Only one directive command can be written in one line.
  - Comments can be written in the directive command line.
- You cannot write a directive command and a mnemonic in the same line.

### Assembly source line

- This line is where a mnemonic is written.
  - Comments can be written in the assembly source line.
  - A label name can be written at the beginning of the assembly source line.
- You cannot write two or more mnemonics in one line.
- You cannot write a directive command and a mnemonic in the same line.

### Label definition line

- This line is where only a label is written.

### Comment line

- This line is where only a comment is written.

### Blank line

- This line contains only space, tab, or line feed code.

### Rules for Writing Lines

#### Separation of lines

Lines are separated by the line feed character, and an interval from a character immediately after a line feed character to the next line feed character is assumed to be one line.

#### Length of line

Up to 255 characters can be written in one line. The as308 assembler does not process the characters in any line exceeding this limit.

- When writing lines of statements, make sure that your description is entered within each line.

The following describes rules on each type of line you need to follow when writing statements.

### Directive command line

#### Function

- Directive command of assembler can be written in this line.

#### Rules for writing

- Always be sure to insert a space or tab between the directive command and its operand.
- When writing multiple operands, always be sure to insert a comma (,) between each operand.
- A space or tab can be inserted between the operand and comma.
- Some directive commands are not accompanied by an operand.
- Directive commands can be written starting immediately from the top of a line.
- A space or tab can be inserted at the beginning of a directive command.

- When writing a comment in the directive command line, insert a semicolon (;) after the directive command and operand and write your comment in columns following the semicolon.
- Comments are output to an assembler list file.
- The as308 assembler processes anything written in columns after the semicolon (;) as a comment. Consequently, the assembler does not generate code for the mnemonics and directive commands written in columns after the semicolon. Therefore, be careful with the position where you enter the semicolon. If a semicolon is enclosed with double quotations (") or single quotation ('), AS308 does not assume it to be the first character of a comment.
- A space or tab can be inserted between a directive command's operand and a comment.

### Description example

```

.SECTION    area,DATA
.ORG    00H
sym .EQU 0
work:    .BLKB 1
.ALIGN
.PAGE "newpage"
.ALIGN           ; Comment

```



## Assembly source line

Refer to the "M16C/80 Series Software Manual" for details on how to write mnemonics. Here, the following explains rules you need to follow to write the assembly source lines that can be processed by as308.

### Function

- Mnemonics available for the M16C/80 Series can be written in this line.

### Rules for writing

- Always be sure to insert a space or tab between the mnemonic and its operand.
- When writing multiple operands, always be sure to insert a comma (,) between each operand.
- A space or tab can be inserted between the operand and comma.
- Some mnemonics are not accompanied by an operand.
- Mnemonics can be written starting immediately from the top of a line.
- A space or tab can be inserted at the beginning of an assembly source line.
- When defining a label in the assembly source line, always be sure to write the label name in columns preceding the mnemonic.
- Be sure to enter a colon before and after the label name.
- A space or tab can be inserted between the label name and the mnemonic.
- When writing a comment in the assembly source line, insert a semicolon (;) after the mnemonic and operand and write your comment in columns following the semicolon.

- Comments are output to an assembler list file.
- The as308 assembler does not generate code for the mnemonics or directive commands written in columns after the semicolon. Therefore, be careful with the position where you enter the semicolon. If a semicolon is enclosed with double quotations (") or single quotation ('), AS308 does not assume it to be the first character of a comment.
- A space or tab can be inserted between a mnemonic's operand and a comment.

### Description example

```
MOV.W #0,R0
RTS
main:    MOV.W #0,A0
RTS                ; End of subroutine
```

## Label definition line

### Function

- Any desired name can be written in this line.

### Rules for writing

- Always be sure to enter the colon (:) immediately after a label name.
- Do not write anything between the label name and the colon (:).
- Label names can be written starting immediately from the top of a line.
- A space or tab can be inserted at the beginning of a line.
- When writing a comment in the label definition line, insert a semicolon (;) after the directive command and operand and write your comment in columns following the semicolon.
- Comments are output to an assembler list file.

■ The as308 assembler does not generate code for the mnemonics or directive commands written in columns after the semicolon. Therefore, be careful with the position where you enter the semicolon. If a semicolon is enclosed with double quotations (") or single quotation ('), AS308 does not assume it to be the first character of a comment.

- A space or tab can be inserted between a label and a comment.

### Description example

```
start:
label:   .BLKB 1
main:   nop
loop:   ; Comment
```

## Comment line

### Function

- Any desired character string can be written in this line.

### Rules for writing

- Always be sure to insert a semicolon (;) at the beginning of a comment.
- A space or tab can be inserted at the beginning of a comment.
- Any desired characters can be written in a comment.

### Description example:

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

## Blank line

### Function

- Nothing apparently is written in this line.

### Rules for writing

- Lines can be entered that do not contain any meaningful characters as may be necessary to improve the legibility of your source program.
- No characters other than the space, tab, return, and line feed characters can be written in a blank line.

### Description example:

```
loop:
:
JMP    loop

JSR    sub1
```

## Line concatenation

- If a line is ended with "\," the next line is concatenated to the position where the "\" is written.
  - A comment can be written in a line where "\" is written. However, no comment is output in the result of concatenation.
  - If an error occurs in a line where "\" is written, the error is output in the last line concatenated.
- The upper limit for the maximum number of characters in all lines that are concatenated is 512 characters. However, this limit does not include the spaces and tabs at the beginning of concatenated lines.
- If a "\" is written immediately after a 2-byte code character, it may be mistaken for "\"." So be careful.
- Description examples for line concatenation and concatenation results are shown below.

Example 1:

```
.BYTE 1,\
      2, \
      3  \
      ,4
```

Concatenation result

```
.BYTE 1,2, 3 ,4
```

## Example 2:

```
.BYTE 1,\\ ;comment  
      2,  ;comment \\  
      3   ;comment
```

## Concatenation result

```
.BYTE 1,2, ;comment  
      3   ;comment
```

## Example 3:

```
.BYTE 1,\\  
      2,\\  
      3,  \\  
      4
```

## Concatenation result

```
.BYTE 1,2,3, 4
```

## Operands

Operands can be written in a mnemonic or directive command to indicate the object to be operated on by that instruction. There are following types of operands.

■ Some instructions do not have an operand. If you want to know whether or not the instruction has an operand, please refer to the rules for writing each command.

- **Numeric value**  
A numeric value includes an integral and a floating- point number.
- **Name**  
A label name and symbol name can be used.
- **Expression**  
An expression with its terms containing a numeric value and a name can be entered.
- **Character string**  
Characters or a character string can be handled as ASCII code.

## Rules for Writing Operands

### Position to write an operand

Always be sure to insert a space or tab between the operand and the instruction that has the operand.

The following describes rules on each type of operand you need to follow when writing an operand.

## Numeric value

A numeric value includes an integral and a floating- point number.

### Integer

An integer can be written in decimal, hexadecimal, binary, or octal notation. The table below shows how to write each type of integer.

- **Binary**

Write a number using numerals 0 to 1 and add 'B' or 'b' at the end of the number.

Example)

10010001B

10010001b

- **Octal**

Write a number using numerals 0 to 7 and add 'O' or 'o' at the end of the number.

Example)

60702O

60702o

- **Decimal**

Write a number using numerals 0 to 9.

Example)

9423

- **Hexadecimal**

Write a number using numerals 0 to 9 and alphabets A to F and add 'H' or 'h' at the end of the number. However, if the number begins with an alphabet, be sure to add a zero '0' at the beginning of the number.

Example)

0A5FH

5FH

0a5fh

5fh



## Floating-point number

The following range of values can be entered that are represented by a floating-point number:

Float (32 bits long):  $1.17549435 \times 10^{-38}$  to  $3.40282347 \times 10^{38}$

Double (64 bits long):  $2.2250738585072014 \times 10^{-308}$  to  $1.7976931348623157 \times 10^{308}$

- Floating-point numbers can only be entered for the operands of directive commands ".DOUBLE" and ".FLOAT".

Example:

|         |                       |
|---------|-----------------------|
| 3.4E35  | 3.4×10 <sup>35</sup>  |
| 3.4e-35 | 3.4×10 <sup>-35</sup> |
| -.5E20  | -0.5×10 <sup>20</sup> |
| 5e-20   | 5.0×10 <sup>-20</sup> |

## Expression

An expression consisting of a combination of numeric value, name, and operator can be entered.

- A space or tab can be inserted between the operator and numeric value.
- Multiple operators can be used in combination.
- When writing an expression as a symbol value, make sure that the value of the expression will be a valid value when your program is assembled.
- The range of values that derive from an expression as a result of operation is -2147483648 to 2147483648.

- Even if the operation results in exceeding the range of -2147483648 to 2147483648, the assembler does not care whether it is an overflow or underflow.

- Floating-point numbers cannot be written in an expression.

- Character constants cannot be used in any terms of an expression.

## Operators

The table below lists the operators that can be written in as308's source programs.

- When writing operators "SIZEOF" and "TOPOF", always be sure to insert a space or tab between the operator and operand.
  
- Relational operators can only be written in the operand of directive commands ".IF" and ".ELIF".)

### *Unary operators*

---

|        |                                                                   |
|--------|-------------------------------------------------------------------|
| +      | Handles value that follows as a positive value.                   |
| -      | Handles value that follows as a negative value.                   |
| ~      | Logically NOT's value that follows.                               |
| SIZEOF | Handles section size(bytes) specified in operand as value.        |
| TOPOF  | Handles start address of section specified in operand as a value. |

---

*Binary operators*

---

|    |                                                                                             |
|----|---------------------------------------------------------------------------------------------|
| +  | Adds values on left and right sides of operand together.                                    |
| -  | Subtracts value on right side of operand from value on left side.                           |
| *  | Multiplies values on left and right sides of operand together.                              |
| /  | Divides value on left side of operand by value on right side.                               |
| %  | Handles remainder derived by dividing value on left side of operand by value on right side. |
| >> | Bit shifts value on left side operand to right as many times as the value on right side.    |
| << | Bit shifts value on left side operand to left as many times as the value on right side.     |
| &  | Logically OR's values on left and right side of operand for each bit.                       |
|    | Logically AND's values on left and right sides of operand for each bit.                     |
| ^  | Exclusive OR's values on left and right sides of operand for each bit.                      |

---

*Relational operators*

---

|    |                                                                                                                                                                                  |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| >  | Evaluates that value on left side of operator is greater than value on right side. This operator can only be written in operand of directive commands .IF and .ELIF.             |
| <  | Evaluates that value on right side of operator is greater than value left side. This operator can only be written in operand of directive commands .IF and .ELIF.                |
| >= | Evaluates that value on left side of operator is equal to or greater than value on right side. This operator can only be written in operand of directive commands .IF and .ELIF. |
| <= | Evaluates that value on right side of operator is equal to or greater than value on left side. This operator can only be written in operand of directive commands .IF and .ELIF. |
| == | Evaluates that value on left side and right side of operator are equal. This operator can only be written in operand of directive commands .IF and .ELIF.                        |
| != | Evaluates that value on left side and right side of operator are not equal. This operator can only be written in operand of directive commands .IF and .ELIF.                    |

---

Operators to priorities operation

- ( ) Operation enclosed with ( ) is performed first before any other operation. If one expression contains multiple parentheses, leftmost pair is given priority. Parenthesized operations can be nested.

**Operation Priority in Expression**

The as308 assembler follows the order of priority shown below as it performs arithmetic operation on the expression written in an operand and handles the value resulting from this operation as an operand value.

- 1 Operation is performed in order of operator priorities, highest priority first. Operator priorities are listed in the table below. The smaller the value shown in this table, the greater the priority.
- 2 Operators of the same priority are operated on sequentially beginning from the left side.
- 3 The priority of operation can be changed by enclosing a given operator with parentheses.

| <i>Priority</i> | <i>Type Operator</i>        | <i>Operator</i>         |
|-----------------|-----------------------------|-------------------------|
| 1               | Operator to change priority | (, )                    |
| 2               | Unary operator              | +, -, ~, sizeof, sizeof |
| 3               | Binary operator 1           | *, /, %                 |
| 4               | Binary operator 2           | +, -                    |
| 5               | Binary operator 3           | >>, <<                  |
| 6               | Binary operator 4           | &                       |
| 7               | Binary operator 5           | !, ^                    |
| 8               | Relational operator         | >, <, >=, <=, ==, !=    |

## Expression and Its Value

The following shows a description example of an expression and the value that results from operations performed by as308.

| <i>Expression</i> | <i>Result of operation</i> |
|-------------------|----------------------------|
| $2+6/2$           | 5                          |
| $(2+6)/2$         | 4                          |
| $1<<3+1$          | 16                         |
| $(1<<3)+1$        | 9                          |
| $3*2\%4/2$        | 1                          |
| $(3*2)\%(4/2)$    | 0                          |
| $8 4/2$           | 10                         |
| $(8 4)/2$         | 6                          |
| $8\&8/2$          | 0                          |
| $(8\&8)/2$        | 4                          |
| $6*-3$            | -18                        |
| $-(6*-3)$         | 18                         |
| $-6*-3$           | 18                         |

## Character String

A character string can be entered in the operand of some directive commands. This character string can be comprised of 7-bit ASCII code characters.

When writing a character string in the operand of a directive command, be sure to enclose it with single or double quotations unless otherwise specified.

Example:

```
"string"
'string'
```

# Directive Commands

AS308 allows you to write directive commands in addition to the M16C/80 series mnemonics in the source programs that can be assembled by AS308. There are following types of directive commands available.

- **Address control directive commands**  
These commands allow you to specify address determination when assembling the source program.
- **Assemble control directive commands**  
These commands allow you to specify how operation is executed by as308.
- **Link control directive commands**  
These commands allow you to define information necessary to control address relocation.
- **List control directive commands**  
These commands allow you to control the format of list files generated by as308.
- **Branch optimization control directive commands**  
These commands allow you to specify that as308 selects the most suitable branch instruction.
- **Conditional assemble control directive commands**  
These commands allow you to select blocks for which code is generated according to conditions set when assembling the source program.
- **Extended function directive commands**  
These commands allow you to control the operations that are not listed above.
- **Directive commands output by M16C/80 series tool software**  
These directive commands and operands all are output by the M16C/80 series tool software.

■ The directive commands output by the M16C/80 series tool software cannot be written in a source program by the user.

## List of Directive Commands

The table below lists the directive commands available with AS308. The following pages explain rules for writing directive commands for each type of directive command.

### Address control

|         |                                           |
|---------|-------------------------------------------|
| .ORG    | Declares address.                         |
| .BLKB   | Allocates RAM area in units of 1 bytes.   |
| .BLKW   | Allocates RAM area in units of 2 bytes.   |
| .BLKA   | Allocates RAM area in units of 3 bytes.   |
| .BLKL   | Allocates RAM area in units of 4 bytes.   |
| .BLKF   | Allocates RAM area in units of 4 bytes.   |
| .BLKD   | Allocates RAM area in units of 8 bytes.   |
| .BYTE   | Stores data in ROM in 1-byte length.      |
| .WORD   | Stores data in ROM in 2-byte length.      |
| .ADDR   | Stores data in ROM in 3-byte length.      |
| .LWORD  | Stores data in ROM in 4-byte length.      |
| .FLOAT  | Stores data in ROM in 4-byte length.      |
| .DOUBLE | Stores data in ROM in 8-byte length.      |
| ALIGN   | Corrects odd addresses to even addresses. |

### Assemble control

|          |                                                                  |
|----------|------------------------------------------------------------------|
| .EQU     | Defines symbol.                                                  |
| .BTEQU   | Defines bit symbol.                                              |
| .END     | Declares end of assemble source.                                 |
| .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 | Reads file into specified position.                              |

**Link control**

|          |                                              |
|----------|----------------------------------------------|
| .SECTION | Defines section name.                        |
| .GLB     | Specifies global label.                      |
| .BTGLB   | Specifies global bit symbol.                 |
| .VER     | Transfers specified information to map file. |

**List control**

|       |                                                               |
|-------|---------------------------------------------------------------|
| .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. |

**Branch instruction optimization control**

|       |                                                                              |
|-------|------------------------------------------------------------------------------|
| .OPTJ | Controls optimization of branch instruction and subroutine call instruction. |
|-------|------------------------------------------------------------------------------|

**Extended Function Directive Commands**

|         |                                                                                              |
|---------|----------------------------------------------------------------------------------------------|
| .ASSERT | Outputs a character string written in the operand to a standard error output device or file. |
| ?       | Specifies defining and referencing a temporary label.                                        |
| ..FILE  | Indicates the assembly source file name being processed by as308.                            |
| @       | Concatenates character strings entered before and after @ into a single character string.    |

**Conditional Assemble Control**

|        |                                                                                   |
|--------|-----------------------------------------------------------------------------------|
| .IF    | Indicates the beginning of a conditional assemble block. Conditions are resolved. |
| .ELIF  | Resolves the second and the following conditions.                                 |
| .ELSE  | Indicates the beginning of a block to be assembled.                               |
| .ENDIF | Indicates the end of a conditional assemble block.                                |



**Macro directive commands**

|          |                                                       |
|----------|-------------------------------------------------------|
| .MACRO   | Defines macro name.Indicates beginning of macro body. |
| .EXITM   | Stops expansion of macro body.                        |
| .LOCAL   | Declares local label in macro.                        |
| .ENDM    | Indicates end of macro body.                          |
| .MREPEAT | Indicates beginning of repeat macro body.             |
| .ENDR    | Indicates end of repeat macro body.                   |

**Macro symbols**

|           |                                                         |
|-----------|---------------------------------------------------------|
| ..MACPARA | Indicates number of actual parameter of macro call.     |
| ..MACREP  | Indicates how many times repeat macro body is expanded. |

**Character string functions**

|         |                                                                                                            |
|---------|------------------------------------------------------------------------------------------------------------|
| .LEN    | Indicates length of specified character string.                                                            |
| .INSTR  | Indicates start position of specified character string in specified character string.                      |
| .SUBSTR | Extracts specified number of characters from specified character string beginning with specified position. |

## ..FILE

Indicates the assembly source file name being processed by as308.

### Function

- This command expands a file name into the one that is being processed by as308 (i.e., assembly source file or include file).
- 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.

### Description format

```
..FILE
```

### Rules for writing command

- This command can be written in the operands of directive commands ".ASSERT" and ".INCLUDE".

### Description example

```
.ASSERT "sample" > ..FILE
```

If the assembly source file name is "sample.a30", a message is output to the "sample" file.

```
.INCLUDE ..FILE@.inc
```

If the assembly source file name is "sample.a30", the "sample.inc" file is included.

```
.INCLUDE "sample" > ..FILE@.mes
```


If the above line is written in "incl.inc" that is included with the "sample.a30" file, a character string normally is output to "incl.mes".

If command option (-F) is specified, a character string is output to the "sample.mes" file.

## ..

Indicates number of actual parameter of macro call

### Function

- This command indicates the number of macro call actual parameters.
  - This command can be written in the body of a macro definition defined by ".MACRO".
-  If this command is written outside the macro body defined by ".MACRO", its value is made 0.

### Description format

```
..
```

### Rules for writing command

- This directive command can be written as a term of an expression.

**Description example**

- The assembler checks the number of macro actual parameters as it executes conditional assemble.

```
      .GLB      mem
name .MACRO    f1,f2
      .IF      .MACPARA == 2
      ADD      f1,f2
      .ELSE
      ADD      R0,f1
      .ENDIF
      .ENDM
      :
name      mem
      :
      .ELSE
      ADD      R0,mem
      .ENDIF
      .ENDM
```

## ..MACREP

Indicates how many times repeat macro body is expanded

### Function

- 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".
- If this command is written outside the macro body, its value is made 0.
- This command can be written in the conditional assemble operand.

### Description format

```
..MACREP
```

### Rules for writing command

- This directive command can be written as a term of an expression.

**Description example**

```

.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
.ENDM

```

## .ADDR

Stores data in ROM in 3-byte length


### Function

- This command stores 3-byte long fixed data in ROM.
- Label can be defined at the address where data is stored.

### Description format

```
.ADDR      (numeric value)
(name:)   .ADDR      (numeric value)
```

### Rules for writing command

- Write an integral value in the operand.
  - Always be sure to insert space or tab between the directive command and the operand.
  - A symbol can be written in the operand.
  - An expression can be written in the operand.
  - When writing multiple operands, separate them with a comma (,).
  - A character or a string of characters can be written in the operand after enclosing it with single quotations (') or double quotations ("). In this case, data is stored in ASCII code representing the characters.
-  The length of a character string you can write in the operand is less than three characters.
- When defining a label, be sure to write the label name before the directive command.
  - Always be sure to insert a colon (:) after the label name.



### Description example

```
.SECTION    value,ROMDATA
.ADDR 1
.ADDR "dat", "a"
.ADDR symbol
.ADDR symbol+1
.ADDR 1,2,3,4,5
.END
```

|             |    |
|-------------|----|
| .ADDR 1     | 01 |
|             | 00 |
|             | 00 |
| .ADDR "dat" | 74 |
|             | 61 |
|             | 64 |
| .ADDR "a"   | 61 |
|             | 00 |
|             | 00 |

## .ALIGN

Corrects odd addresses to even addresses

### Function

- This command corrects the address to an even address at which code in the line immediately following description of the command is stored.
- If the section type is CODE or ROMDATA, the NOP code (04H) is written into an address that has been emptied as a result of address correction.
- If the section type is DATA, the address value is incremented by 1.
- Address correction is not performed if the address in which this command is written is an even address.

### Description format

```
.ALIGN
```

### Rule for writing command

- This directive command can be written in a section that falls under the conditions below:

A relative-attribute section in which address correction is directed when defining the section

```
.SECTION program, CODE, ALIGN
```

An absolute-attribute section

```
.SECTION program, CODE
.ORG 0e000H
```

### Description example

```

.SECTION    program, CODE, ALIGN
MOV.W #0, R0
.ALIGN
.END

.SECTION    program, CODE
.ORG 0f000H
MOV.W #0, R0
.ALIGN
.END

```

| Source   |                       | Address | Code   |                              |
|----------|-----------------------|---------|--------|------------------------------|
| .SECTION | count, ROMDATA, ALIGN |         |        |                              |
| .ADDR    | 1                     | 00000   | 010000 |                              |
| .ALIGN   |                       | 00003   | 04     | NOP code is inserted.        |
| .SECTION | ram, DATA, ALIGN      |         |        |                              |
| .BLKA    | 1                     | 00000   |        |                              |
| .ALIGN   |                       | 00003   |        | Address is incremented by 1. |
| .BLKB    | 1                     | 00004   |        |                              |
| .END     |                       |         |        |                              |

## .ASSERT

Output a character string written in the operand

### Function

- 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.
- If the file name does not have directory specification, the assembler generates the file in the current directory.

### Description format

```
.ASSERT  "(character string)"
.ASSERT  "(character string)" > (file name)
.ASSERT  "(character string)" >> (file name)
```

### Rules for writing command

- 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.

### Description example

```
.ASSERT "string" > sample.dat
```

Message is output to file sample.dat.

```
.ASSERT "string" >> sample.dat
```

Message is added to file sample.dat.

```
.ASSERT "string" > ..FILE
```

Message is output to a file bearing the same name as the currently processed file except the extension.

## .BLKA

Allocates RAM area in units of 3 bytes

### Function

- This command allocates a specified bytes of RAM area in units of 3 bytes.
- Label name can be defined at the allocated RAM address.

### Description format

```
.BLKA      (numeric value)
(name:)    .BLKA      (numeric value)
```

### Rules for writing command

- This directive command must always be written in a DATA-type section. Section types can be made the DATA type simply by writing ",DATA" following the section name when you define a section.
- Always be sure to insert space or tab between the directive command and the operand.
- Write an integral value in the operand.
- A symbol can be written in the operand.
- An expression can be written in the operand.
- The expression in the operand must have its values determined when assembling the source program.
- When defining a label name in the allocated area, be sure to write the label name before the directive command. Always be sure to insert a colon (:) after the label name.

### Description example

```
symbol    .EQU      1
          .SECTION  area,DATA
work1:    .BLKA 1
work2:    .BLKA symbol
          .BLKA symbol+1
```

## .BLKB

Allocates RAM area in units of 1 bytes

### Function

- This command allocates a specified bytes of RAM area in units of 1 byte.
- Label name can be defined at the allocated RAM address.

### Description format

```
.BLKB      (numeric value)
(name:)   .BLKB      (numeric value)
```

### Rules for writing command

- This directive command must always be written in a DATA-type section. Section types can be made the DATA type simply by writing ",DATA" following the section name when you define a section.
- Always be sure to insert space or tab between the directive command and the operand.
- Write an integral value in the operand.
- A symbol can be written in the operand.
- An expression can be written in the operand.
- The expression in the operand must have its values determined when assembling the source program.
- When defining a label name in the allocated area, be sure to write the label name before the directive command. Always be sure to insert a colon (:) after the label name.

### Description example

```
symbol    .EQU      1
          .SECTION  area,DATA
work1:    .BLKB 1
work2:    .BLKB symbol
          .BLKB symbol+1
```

## .BLKD

Allocates RAM area in units of 8 bytes

### Function

- This command allocates a specified bytes of RAM area in units of 8 bytes.
- Label name can be defined at the allocated RAM address.

### Description format

```
.BLKD      (numeric value)
(name:)   .BLKD      (numeric value)
```

### Rules for writing command

- This directive command must always be written in a DATA-type section. Section types can be made the DATA type simply by writing ",DATA" following the section name when you define a section.
- Always be sure to insert space or tab between the directive command and the operand.
- Write an integral value in the operand.
- A symbol can be written in the operand.
- An expression can be written in the operand.
- The expression in the operand must have its values determined when assembling the source program.
- When defining a label name in the allocated area, be sure to write the label name before the directive command. Always be sure to insert a colon (:) after the label name.

### Description example

```
symbol    .EQU      1
          .SECTION  area,DATA
work1:    .BLKD 1
work2:    .BLKD symbol
          .BLKD symbol+1
```



## .BLKF

Allocates RAM area in units of 4 bytes

### Function

- This command allocates a specified bytes of RAM area in units of 4 bytes.
- Label name can be defined at the allocated RAM address.

### Description format

```
.BLKF      (numeric value)
(name:)   .BLKF      (numeric value)
```

### Rules for writing command

- This directive command must always be written in a DATA-type section. Section types can be made the DATA type simply by writing ",DATA" following the section name when you define a section.
- Always be sure to insert space or tab between the directive command and the operand.
- Write an integral value in the operand.
- A symbol can be written in the operand.
- An expression can be written in the operand.
- The expression in the operand must have its values determined when assembling the source program.
- When defining a label name in the allocated area, be sure to write the label name before the directive command. Always be sure to insert a colon (:) after the label name.

### Description example

```
symbol    .EQU      1
          .SECTION  area,DATA
work1:    .BLKF 1
work2:    .BLKF symbol
          .BLKF symbol+1
```

## .BLKL

Allocates RAM area in units of 4 bytes

### Function

- This command allocates a specified bytes of RAM area in units of 4 bytes.
- Label name can be defined at the allocated RAM address.

### Description format

```
.BLKL      (numeric value)
(name:)   .BLKL      (numeric value)
```

### Rules for writing command

- This directive command must always be written in a DATA-type section. Section types can be made the DATA type simply by writing ",DATA" following the section name when you define a section.
- Always be sure to insert space or tab between the directive command and the operand.
- Write an integral value in the operand.
- A symbol can be written in the operand.
- An expression can be written in the operand.
- The expression in the operand must have its values determined when assembling the source program.
- When defining a label name in the allocated area, be sure to write the label name before the directive command. Always be sure to insert a colon (:) after the label name.

### Description example

```
symbol    .EQU      1
          .SECTION  area,DATA
work1:    .BLKL 1
work2:    .BLKL symbol
          .BLKL symbol+1
```

## .BLKW

Allocates RAM area in units of 2 bytes

### Function

- This command allocates a specified bytes of RAM area in units of 2 bytes.
- Label name can be defined at the allocated RAM address.

### Description format

```
.BLKW      (numeric value)
(name:)   .BLKW      (numeric value)
```

### Rules for writing command

- This directive command must always be written in a DATA-type section. Section types can be made the DATA type simply by writing ",DATA" following the section name when you define a section.
- Always be sure to insert space or tab between the directive command and the operand.
- Write an integral value in the operand.
- A symbol can be written in the operand.
- An expression can be written in the operand.
- The expression in the operand must have its values determined when assembling the source program.
- When defining a label name in the allocated area, be sure to write the label name before the directive command. Always be sure to insert a colon (:) after the label name.

### Description example

```
symbol    .EQU      1
          .SECTION  area,DATA
work1:    .BLKW 1
work2:    .BLKW symbol
          .BLKW symbol+1
```

## .BTEQU

Defines bit symbol

### Function

- 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.
- Bit symbols can be used in symbolic debug.
- Bit symbols can be specified as global.

### Description format

```
(name) .BTEQU (bit position), (address value)
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- Separate between the bit position and the bit's memory address with a comma as you enter them.
- Always be sure to write the bit position first and then the address value.
- An integer in the range of 0 to 65535 can be written to indicate the bit position.
- Always make sure that the value you specify for the bit position is determined when assembling the source program.
- A symbol can be written to specify the address value of an operand.
- A label or symbol that is indeterminate when assembled can be written to specify the address value of an operand.

- 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 can be written in the operand.
- However, 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.
- An expression can be written in the operand.

#### Description example

```
bit0 .BTEQU      0,0
bit1 .BTEQU      1,flag
bit2 .BTEQU      2,flag+1
bit3 .BTEQU      one,flag
bit4 .BTEQU      one+one,flag
```

## .BTGLB

Specifies global bit symbol

### Function

- This command declares that the bit symbols specified with it are global symbols.
- If any bit symbols specified with this directive command are not defined within the file, the assembler processes them assuming that they are defined in an external file.
- If the bit symbols specified with this directive command are defined in the file, the assembler processes them to be referencible from an external file.

### Description format

```
.BTGLB      (bit symbol name)
.BTGLB      (bit symbol name) [,(bit symbol name)...]
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- Write a bit symbol name in the operand that you want to be a global symbol.
- No bit symbols can be specified for external reference that are defined by a symbol that is indeterminate when assembled.
- When specifying multiple bit symbol names in the operand, separate each symbol name with a comma (,) as you write them.

### Description example

```
.BTGLB      flag1,flag2,flag3
.BTGLB      flag4
.SECTION    program
BCLR       flag1
```

## .BYTE

Stores data in ROM in 1-byte length

### Function

- This command stores 1-byte long fixed data in ROM.
- Label can be defined at the address where data is stored.

### Description format

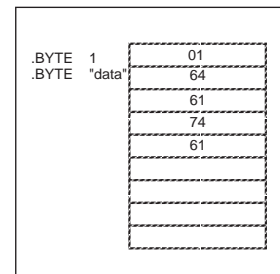
```
.BYTE      (numeric value)
(name:)    .BYTE      (numeric value)
```

### Rules for writing command

- Write an integral value in the operand.
- Always be sure to insert space or tab between the directive command and the operand.
- A symbol can be written in the operand.
- An expression can be written in the operand.
- When writing multiple operands, separate them with a comma (,).
- A character or a string of characters can be written in the operand after enclosing it with single quotations (') or double quotations ("). In this case, data is stored in ASCII code representing the characters.
- When defining a label, be sure to write the label name before the directive command.
- Always be sure to insert a colon (:) after the label name.

### Description example


```
.SECTION    value,ROMDATA
.BYTE 1
.BYTE "data"
.BYTE symbol
.BYTE symbol+1
.BYTE 1,2,3,4,5
.END
```



## .DEFINE

Defines string symbol

### Function

- This command defines a character string to a symbol.
  - A symbol can be redefined.
-  The symbols defined by this directive command cannot be specified for external reference.

### Description format

```
(symbol name) .DEFINE (character string)
(symbol name) .DEFINE '(character string)'
(symbol name) .DEFINE "(character string)"
```

### Description rules

- 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.

### Description example

```
                .SECTION    ram,DATA
data1:         .BLKB 1
flag .DEFINE   "#01H, data1"
                .SECTION    program
                CLB         flag
```



## .DOUBLE

Stores data in ROM in 8-byte length

### Function

- This command stores 8-byte long fixed data in ROM.
- Label can be defined at the address where data is stored.

### Description format

```
.DOUBLE (numeric value)
(name:) .DOUBLE (numeric value)
```

### Rules for writing command

- Write a floating-point number in the operand.
- Refer to "Rules for writing operand" for details on how to write a floating-point number in the operand.
- Always be sure to insert space or tab between the directive command and the operand.
- When defining a label, be sure to write the label name before the directive command.
- Always be sure to insert a colon (:) after the label name.

### Description example

```
.DOUBLE 5E2
constant: .DOUBLE 5e2
```

## .ELIF

Resolves the second and the following conditions

### Function

- 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.

### Description format

```
.IF      {conditional expression}  
body  
.ELIF   {conditional expression}  
body  
.ENDIF
```

### Rules for writing command

- Always be sure to write a conditional expression in the operand of this directive command.
- Always be sure to insert space or tab between the directive command and the operand.
- This directive command can be written for multiple instances in one conditional assemble block.

### Description example

```
.IF      TYPE==0  
.byte "Proto Type Mode"  
.ELIF   TYPE>0  
.byte "Mass Production Mode"  
.ELSE  
.byte "Debug Mode"  
.ENDIF
```

## .ELSE

Indicates the beginning of a block to be assembled

### Function

- 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.

### Description format

```
.IF      {conditional expression}
body
.ELSE    {conditional expression}
body
.ENDIF
```

```
.IF      {conditional expression}
body
.ELIF    {conditional expression}
body
.ELSE
body
.ENDIF
```

### Rules for writing command

- This directive command can be written less than once in a conditional assemble block.
- This directive command does not have an operand.

### Description example

```
.IF      TYPE==0
.byte "Proto Type Mode"
.ELIF    TYPE>0
.byte "Mass Production Mode"
.ELSE
.byte "Debug Mode"
.ENDIF
```

## .END

Declares end of assemble source

### Function

- 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.

### Description format

```
.END
```

### Rules for writing command

- There must always be at least one of this directive command in one assembly source file.
- The as308 assembler does not detect errors in the subsequent lines after this directive command either.

### Description example

```
.END
```

## .ENDIF

Indicates the end of a conditional assemble block

### Function

- This command indicates the end of the conditional assemble block.

### Description format

```
.IF      {conditional expression}
body
.ENDIF
```

### Rules for writing command

- 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.

### Description example

```
.IF      TYPE==0
.byte "Proto Type Mode"
.ELIF TYPE>0
.byte "Mass Production Mode"
.ELSE
.byte "Debug Mode"
.ENDIF
```

## .ENDM

Indicates end of macro body

### Function

- This command indicates that the body of one macro definition is terminated here.

### Rules for writing command

- Always make sure that this command corresponds to directive command ".MACRO" as you write it.

### Description format

```
(macro name)      .MACRO
body
.ENDM
```

### Description example

```
lda .MACRO      value
MOV.W #value,A0
.ENDM
:
lda 0
:
MOV.W #0,A0
```

## .ENDR

Indicates end of repeat macro body

### Function

- This command indicates the end of a repeat macro.

### Description format

```
[ (label) : ]      .MREPEAT      (numeric value)
body
.ENDR
```

### Rules for writing command

- Always make sure that this command corresponds to directive command ".MREPEAT" as you write it.

### Description example

```
rep .MACRO      num
    .MREPEAT    num
    .IF         num > 49
    .EXITM
    .ENDIF
nop
.ENDR
.ENDM
:
rep      3
:
nop
nop
nop
```

## .EQU

Defines symbol

### Function

- This command defines a value in the range of signed 32-bit integers (-32768 to 32767) to a symbol.
- Symbolic debug function is made available for use by defining symbols with this directive command.

### Description format

```
(name)      .EQU      (numeric value)
```

### Rules for writing command

- The value that can be defined to a symbol must be determined when assembling the source program.
- Always be sure to insert space or tab between the directive command and the operand.
- A symbol can be written in a symbol-defined operand.
- However, symbol names cannot be entered that are forward referenced.
- An expression can be written in a symbol-defined operand.
- Symbols can be specified as global.

### Description example

```
symbol      .EQU  1
symbol1     .EQU  symbol+symbol
symbol2     .EQU  2
```



## .EXITM

Stop expansion of macro body

### Function

- This command stops expanding the macro body and transfers control to the nearest ".ENDM".

### Description format

```
(macro name) .MACRO
body
.EXITM
body
.ENDM
```

### Rules for writing command

- Make sure that the command is written within the body of a macro definition.

### Description example

```
data1 .MACRO value
      .IF value == 0
      .EXITM
      .ELSE
      .BLKB value
      .ENDIF
      .ENDM
      :
data1 0
      :
      .IF 0 == 0
      .EXITM
      .ENDIF
      .ENDM
```

## .FB

Assigns temporary FB register value

### Function


- This command assigns a provisional FB register value.
- When assembling the source program, the assembler assumes that the FB register value is one that is defined by this directive command as it generates code for the subsequent source lines.
- FB relative addressing mode can be specified in the subsequent lines.
- The assembler generates code in FB relative addressing mode for the mnemonics that use labels defined by directive command ".FBSYM".

### Description format

```
.FB      (numeric value)
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- Always make sure that this command is written in the assembly source file.
- Always be sure to write this command before you use the FB relative addressing mode.
- An integer in the range of 0 to 0FFFFFFH can be written in the operand.

 This directive command only directs the assembler to take on a provisional FB register value and cannot be used to set a value to the actual FB register. To set an FB register value actually, write the following instruction immediately before or after this directive command. Example: LDC #80H,FB

- A symbol can be written in the operand.

### Description example

```
.FB      80H
LDC      #80H,FB
```

## .FBSYM

Selects FB relative displacement addressing mode

### Function

- The assembler selects the FB relative addressing mode for the name specified in the operand of this directive command.
- The assembler selects the FB relative addressing mode for the operand in absolute 16-bit addressing mode that includes the name specified in the operand of this directive command.

### Description format

```
.FBSYM      (name)
.FBSYM      (name) [ , (name) . . . ]
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- Always be sure to set the FB register value with directive command ".FB" before you write this directive command.
- When specifying multiple names, be sure to separate the names with a comma as you write them.
- Be careful that the symbol you specify with this directive command is not a duplicate of the symbol specified by ".SBSYM".

### Description example

```
.FB      80H
LDC      #80,FB
.FBSYM      sym1, sym2
```

## .FLOAT

Stores data in ROM in 4-byte length

### Function

- This command stores 4-byte long fixed data in ROM.
- Label can be defined at the address where data is stored.

### Description format

```
.FLOAT      (numeric value)
(name:)    .FLOAT      (numeric value)
```

### Rules for writing command

- Write a floating-point number in the operand.
- Refer to "Rules for writing operand" for details on how to write a floating-point number in the operand.
- Always be sure to insert space or tab between the directive command and the operand.
- When defining a label, be sure to write the label name before the directive command.
- Always be sure to insert a colon (:) after the label name.

### Description example

```
.FLOAT      5E2
constant:   .FLOAT      5e2
```

## .FORM

Specifies number of columns and lines in 1 page of list file

### Function

- 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 = 140.

### Description format

```
.FORM      (number of lines), (number of columns)
.FORM      (number of lines)
.FORM      ,(number of columns)
```

### Rules for writing command

- 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.
- Symbols cannot be used that are forward referenced.
- 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.

### Description example

```
.FORM 20,80  
.FORM 60  
.FORM ,100  
.FORM line,culmn
```

## .GLB

Specifies global label

### Function

- This command declares that the labels and symbols specified with it are global.
- If any labels or symbols specified with this directive command are not defined within the file, the assembler processes them assuming that they are defined in an external file.
- If the labels or symbols specified with this directive command are defined in the file, the assembler processes them to be referencible from an external file.

### Description format

```
.GLB      (name)
.GLB      (name) [ ,(name)... ]
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- Write a label name in the operand that you want to be a global label.
- Write a symbol name in the operand that you want to be a global symbol.
- When specifying multiple symbol names in the operand, separate each symbol name with a comma (,) as you write them.

### Description example

```
.GLB  name1,name2,name3
.GLB  name4
.SECTION  program
MOV.W #0,name1
```

## .IF

### Conditional assemble control

#### Function

- This command indicates the beginning of a conditional assemble block.
- 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.
- Any instructions that can be written in a as308 source program can be written in the conditional assemble block.

#### Description format

```
.IF      {conditional expression}  
body  
.ENDIF
```

#### Rules for writing command

- Always be sure to write a conditional expression in the operand of this directive command.
- Always be sure to insert space or tab between the directive command and the operand.

#### Function of conditional expression

- Conditional assemble is performed based on the result of the conditional expression.



### Rules for writing conditional expression

- Only one conditional expression can be written in the operand of the directive command.
- Always be sure to write a relational operator in the conditional expression.
- The operators listed below can be used.

| <i>Relational operators</i> | <i>Contents</i>                                                                         |
|-----------------------------|-----------------------------------------------------------------------------------------|
| >                           | True if value on left side of operator is greater than value on right side.             |
| <                           | True if value on right side of operator is greater than value on left side.             |
| >=                          | True if value on left side of operator is equal to or greater than value on right side. |
| <=                          | True if value on right side of operator is equal to or greater than value on left side. |
| ==                          | True if values on left and right sides of operator are equal.                           |
| !=                          | True if values on left and right sides of operator are not equal                        |

- Arithmetic operation of a conditional expression is performed in signed 32 bits.

**■** The assembler does not care whether the operation has resulted in overflow or underflow.

- A symbol can be written in the left and right sides of the relational operator.

**■** Symbols cannot be forward referenced (only the symbols that are defined after this directive command are referenced). Forward referenced symbols or undefined symbols written here are assumed to be 0 in value as the assembler resolves the conditional expression.

- An expression can be written on the left and right sides of the relational operator. To write an expression, follow the "rules for writing expression" in Section 1, "Rules for Writing Program".
- A character string can be written on the left and right sides of the relational operator. Always be sure to enclose the character string with single quotations (') or double quotations (") as you write it. Which character string is larger or smaller than the other is resolved by the value of character code.

```
"ABC" < "CBA"
-> 414243 < 434241; therefore, condition is true.
```

```
"C" < "A"
-> 43 < 41; therefore, condition is false.
```

- Space or tab can be written before or after the relational operator.
- A conditional expression can be written in the operands of directive commands ".IF" and ".ELIF".

### Description example of conditional expression

```
sym<1
sym < 1
sym+2 < data1
sym+2 < data1+2
'smpl'==name
```

### Description example

```
.IF TYPE==0
.byte "Proto Type Mode"
.ELIF TYPE>0
.byte "Mass Production Mode"
.ELSE
.byte "Debug Mode"
.ENDIF
```

# .INCLUDE

Reads file into specified position

## Function

- This command reads the content of a specified file into a line of the source program.
- Nesting level of include files is within 9.
- When you describe an include file name with an absolute path name, AS308 searches the directory described in the operand field for a file. An error occurs if no file can be found in the directory.
- When you describe a include file with a relative path name or a file name alone in the operand field of the inclusion-directing instruction:
  - 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 AS308: AS308 searches for a file name designated by the inclusion-directing instruction.  
In an instance in which a directory is designated for a file name designated in the command line at the time of starting up AS308: AS308 searches for a file name resulting from adding a directory name specified in the command line to a file name specified by the inclusion-directing instruction.
  - 2 AS308 searches the directory designated by the command option -I.
  - 3 AS308 searches the directory set in the environment variable INC79.

## Description format

```
.INCLUDE (file name)
```

## Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
  - Always be sure to write a file extension in the operand file name.
  - A character string that includes directive command "..FILE" or "@" can be written.
- Do not specify INCLUDE the file itself within the include file.

## Description example

```
.INCLUDE initial.a30
```

## .INSTR

Detects specified character string

### Function

- 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.
- 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.

### Description format

```
.INSTR      { "(CS)", "(SC)", (SP) }
.INSTR      { '(CS)', '(SC)', (SP) }
CS=character string
SC=search character string
SP=search start position
```

### Rules for writing command

- Always be sure to enclose the operand with { }.
- Always be sure to write the character string, search character string, and search start position.
- Separate the character string, search character string, and search start position with commas as you write them.
- No space or tab can be inserted before and after the comma.
- 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.
- Kanji and other 8-bit code are not processed correctly. However, the as308 assembler does not output errors.
- Always be sure to enclose the character string with quotations as you write it.
- 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.
- This directive command can be written as a term of an expression.

### Description example

```

top .EQU 1

point_set .MACRO    source,dest,top
point     .EQU     .INSTR{'source','dest',top}
        .ENDM
        :
        point_set  japanese,se,1
        :
point    .EQU 7

```

This example extracts the position (7) of the character string "se" from the beginning (top) of the specified character string (japanese).

## .LEN

Indicates length of specified character string

### Function

- This command indicates the length of the character string that is written in the operand.

### Description format

```
.LEN      {"(character string)" }  
.LEN      {'(character string)'} 
```

### Rules for writing command

- 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 be used to write a character string.
- Kanji and other 8-bit code are not processed correctly. However, the as308 assembler does not output errors.
- Always be sure to enclose the character string with quotations as you write it.
- 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.
- This directive command can be written as a term of an expression.

### Description example

```
bufset .MACRO f1,f2
buffer@f1: .BLKB .LEN{'f2'}
.ENDM
:
bufset 1,Printout_data
bufset 2,Sample
:
buffer1 .BLKB 13
buffer2 .BLKB 6

buf .MACRO f1
buffer: .BLKB .LEN{"f1"}
.ENDM
:
buf 1,data ; data is not expanded.
:
buffer .BLKB 2
```

## .LIST

Controls outputting of line data to list file

### Function

- This command allows you to stop (OFF) outputting lines to the assembler list file.
- Lines in error are output to the list file regardless of whether they are within the list output disabled range.
- This command allows you to start (ON) outputting lines to the assembler list file.
- All lines are output to the list file if you do not specify this directive command.

### Description format

```
.LIST      [ON|OFF]
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- To stop outputting lines, write 'OFF' in the operand.
- To start outputting lines, write 'ON' in the operand.

### Description example

```
.LIST ON
.LIST OFF
```

| Example of source file | Example of assembler list file output |
|------------------------|---------------------------------------|
| MOV.B #0,R0L           | MOV.B #0,R0L                          |
| MOV.B #0,R0L           | MOV.B #0,R0L                          |
| .LIST OFF              | .LIST OFF                             |
| MOV.B #0,R0L           | MOV.B #0,R0                           |
| MOV.B #0,R0L           |                                       |
| MOV.B #0,R0            | ← Line in error → Error message       |
| MOV.B #0,R0L           |                                       |
| MOV.B #0,R0L           |                                       |
| .LIST ON               | .LIST ON                              |
| MOV.B #0,R0L           | MOV.B #0,R0L                          |
| MOV.B #0,R0L           | MOV.B #0,R0L                          |
| MOV.B #0,R0L           |                                       |



## .LOCAL

Declares local label in macro

### Function

- 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.
- 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.

### Description format

```
.LOCAL (label name)[,(label name)...]
```

### Rules for writing command

- Always make sure that this directive command is written within the macro body.
- Always be sure to insert space or tab between this directive command and the operand.
- 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 1, "Rules for Writing Program".
- 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.

- 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.

### Description example

```
name      .MACRO
          .LOCAL      m1;'m1' is the macro local label.
m1:
          NOP
          JMP          m1
          .ENDM
```

## .LWORD

Stores data in ROM in 4-byte length

### Function

- This command stores 4-byte long fixed data in ROM.
- Label can be defined at the address where data is stored.

### Description format

```

                .LWORD    (numeric value)
(name:)        .LWORD    (numeric value)

```

### Rules for writing command

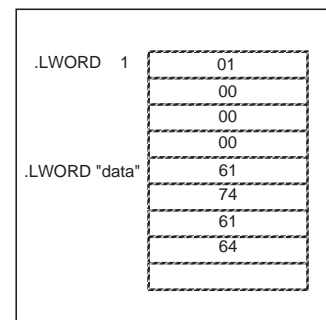
- Write an integral value in the operand.
  - Always be sure to insert space or tab between the directive command and the operand.
  - A symbol can be written in the operand.
  - An expression can be written in the operand.
  - When writing multiple operands, separate them with a comma (,).
  - A character or a string of characters can be written in the operand after enclosing it with single quotations (') or double quotations ("). In this case, data is stored in ASCII code representing the characters.
- The length of a character string you can write in the operand is less than four characters.
- When defining a label, be sure to write the label name before the directive command.
  - Always be sure to insert a colon (:) after the label name.

### Description example

```

.SECTION    value,ROMDATA
.LWORD     1
.LWORD     "data"
.LWORD     symbol
.LWORD     symbol+1
.LWORD     1,2,3,4,5
.END

```



## .MACRO

Defines macro name and beginning of macro body

### Function

- This command defines a macro name.
- This command indicates the beginning of macro definition.

### Description format

- Macro definition

```
(macro name) .MACRO [(formal parameter) [, (formal parameter)...]]  
body  
.ENDM
```

- Macro call

```
(macro name) [(actual parameter)[, (actual parameter)...]]
```

### Rules for writing command

- Always be sure to write a macro name.
- To write a macro name, follow the rules for writing name in Section 1, "Rules for Writing Program".
- Formal parameters can be defined in the operand.
- Always be sure to insert space or tab between this directive command and the macro formal parameter.
- Space or tab can be written between this directive command and the macro name.

### Rules for writing formal parameter

- To write a macro formal parameter name, follow the rules for writing name in Section 1, "Rules for Writing Program".
  - When defining a macro formal parameter, use a name that is unique including nested macro definitions.
  - When defining multiple formal parameters, separate the formal parameters with a comma (,) as you write them.
  - Always make sure that the formal parameters written in the operand of directive command ".MACRO" are written within the macro body.
- 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.
- Up to 80 formal parameters can be entered.
- This means that you can enter up to 80 formal parameters within the range of the number of characters that can be written in one line.

### Rules for writing actual parameter

- Always be sure to insert space or tab between the macro name and the actual parameter.
- Make sure that the actual parameters you write are corresponded one for one to the formal parameters when the macro is called.
- When using a special character to write a actual parameter, be sure to enclose the character with double quotations as you write it.
- Labels, global labels, and symbols can be used to write actual parameters.
- An expression can be entered in a actual parameter.

### Expanding actual parameter

- Formal parameters are replaced with actual parameters sequentially from left to right in the order they are written.
  - If no actual parameter is written in macro call that corresponds to a defined formal parameter, the assembler does not generate code for this formal parameter part.
  - If there are more formal arguments than the actual arguments and some formal arguments do not have the corresponding actual arguments, the assembler does not generate code for this formal argument part.
  - If a formal parameter written in the body is enclosed with single quotations ('), the assembler encloses the corresponding actual parameter with single quotations as it is output.
  - If one actual parameter contains a comma (,) while at the same time the argument is enclosed with parentheses "()", the assembler converts the argument along with its parentheses.
  - If there are more actual parameters than the formal parameters, the assembler does not process the actual parameters that do not have the corresponding formal parameters.
- If the number of actual parameters does not match that of formal parameters, the as308 assembler outputs a warning message.

## 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)'
```

## Description example

```
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
:
mac  word,10,R0
:
      .IF      3=3
          .ELSE
              MOV.W #10,R0
          .ENDIF
      .ENDIF
      .ENDM
```

## .MREPEAT

Indicates beginning of repeat macro body


### Function

- 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.

### Description format

```
[(label):] .MREPEAT (numeric value)
body
.ENDR
```

### Rules for writing command

- Always be sure to write the operand.
  - Always be sure to insert space or tab between this directive command and the operand.
  - A label can be written at the beginning of this directive command.
  - A symbol can be written in the operand.
-  Forward referenced symbols cannot be used here.
- An expression can be written in the operand.
  - Macro definition and macro call can be written in the body.
  - Directive command ".EXITM" can be written in the body.



**Description example**

```
rep .MACRO      num
    .MREPEAT   num
    .IF      num > 49
        .EXITM
    .ENDIF
NOP
    .ENDR
    .ENDM
    :
rep 3
    :
NOP
NOP
NOP
```

## .OPTJ

Controls optimization

### Function

- This command controls optimization of unconditional branch instructions.
- A jump distance can be specified for unconditional branch instructions or subroutine call instructions where the jump distance specifier is omitted and the operand is not subject to optimization processing.
- The specified contents become effective beginning with the line following one in which this directive command is written.
- Optimization specification by this directive command can be entered for multiple instances in one assembly source file.

### Description format

```
.OPTJ      [OFF|ON], [JMPW|JMPA], [JSRW|JSRA]
```

### Rules for writing command

- The following three parameters can be written in the operand of this directive command:
  - 1 Optimization control of branch instruction
  - 2 Selection of unconditional branch instruction excluded from optimization processing
  - 3 Selection of subroutine call instruction excluded from optimization processing

**The following contents can be written in each parameter:**

|   |      |                                                                                                                |
|---|------|----------------------------------------------------------------------------------------------------------------|
| 1 | OFF  | Branch instructions are not optimized.                                                                         |
|   | ON   | Branch instructions are optimized. (Default)                                                                   |
| 2 | JMPW | Unconditional branch instructions not subject to optimization processing are generated with "JMP.W".           |
|   | JMPA | Unconditional branch instructions not subject to optimization processing are generated with "JMP.A". (Default) |
| 3 | JSRW | Subroutine call instructions not subject to optimization processing are generated with "JSR.W".                |
|   | JSRA | Subroutine call instructions not subject to optimization processing are generated with "JSR.W". (Default)      |

- Each parameter can be specified in any desired order.
- Each parameter can be omitted. If any parameter is omitted, the jump distance does not change beginning with the default value or previously specified content.

**Description example**

A combination of operands shown below can be entered:

```
.OPTJ OFF
.OPTJ ON
.OPTJ ON, JMPW
.OPTJ ON, JMPW, JSRW
.OPTJ ON, JMPW, JSRA
.OPTJ ON, JMPA
.OPTJ ON, JMPA, JSRW
.OPTJ ON, JMPA, JSRA
.OPTJ ON, JSRW
.OPTJ ON, JSRA
```

## .ORG

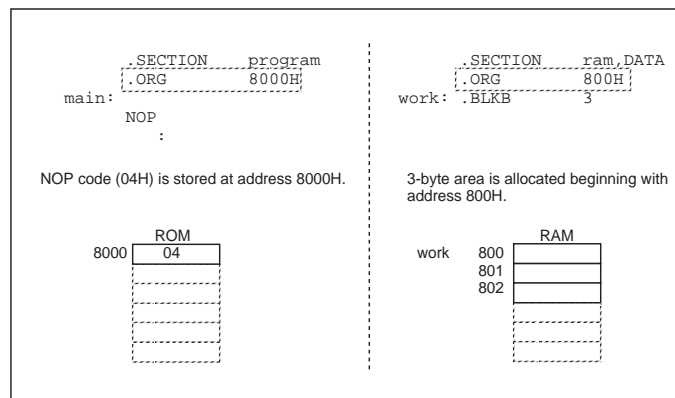
Specifies address value

### Function

- Sections in which this directive command is written are assigned absolute attribute.
- Absolute-attribute sections cannot have their addresses relocated when linking programs.
- The addresses of a section in which this directive command is written take on absolute values.
- The addresses where code is stored for mnemonics that are written in the lines immediately following this directive command are determined.
- The memory addresses to be allocated by an area allocating directive command that is written in the lines immediately following this directive command are determined.

### Description format

`.ORG (numeric value)`



### Rules for writing command

- This directive command must always be written immediately after a section directive command.
- If directive command ".ORG" is not found in the line immediately following description of ".SECTION", the section is assigned relative attribute.
- This directive command cannot be written in relative-attribute sections.
- Always be sure to insert space or tab between the directive command and the operand.
- The values that can be written in the operand are a numeric value in the range of 0 to 0FFFFFFH.
- An expression can be written in the operand. However, this expression must have its values determined when assembling the source program.
- A symbol can be written in the operand. However, this symbol must have its values determined when assembling the source program.
- This directive command can not be written in sections that are specified to be relative attribute.
- This directive command can be written for multiple instances within an absolute-attribute section.

### Description example

```
.SECTION    value,ROMDATA
.ORG    0FF00H
.BYTE "abcdefghijklmnopqrstuvwxy"
.ORG    0FF80H
.BYTE "ABCDEFGHIJKLMNQPQRSTUVWXYZ"
.END
```

The following statement results in an error.

```
.SECTION    value,ROMDATA
.BYTE "abcdefghijklmnopqrstuvwxy"
.ORG    0FF80H
.BYTE "ABCDEFGHIJKLMNQPQRSTUVWXYZ"
```

## .PAGE

Breaks pages at specified position of list file

### Function

- This command causes pages in the assembler list file to break.
  - The character string written in the operand is output to the header section in the new page of the assembler list file.
- The maximum number of characters that can be output to the header is value subtracted 65 from the number of columns in the list file. Use directive command ".FORM" to set the number of columns in the list file.

### Description format

```
.PAGE      "(character string)"  
.PAGE      '(character string)'
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- Enclose the operand with single quotations (') or double quotations (") as you write it.
- The operand can be omitted.

### Description example

```
.PAGE  
.PAGE "strings"  
.PAGE 'strings'
```

## .SB

Assigns temporary SB register value

### Function


- This command assigns a provisional SB register value.
- When assembling the source program, the assembler assumes that the SB register value is one that is defined by this directive command as it generates code for the subsequent source lines.
- SB relative addressing mode can be specified in the subsequent lines.
- The assembler generates code in SB relative addressing mode for the mnemonics that use labels defined by directive command ".SBSYM".

### Description format

```
.SB      (numeric value)
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- Always make sure that this command is written in the assembly source file.
- Always be sure to write this command before you use the SB relative addressing mode.
- AN integer in the range of 0 to 0FFFFH can be written in the operand.

 This directive command only directs the assembler to take on a provisional SB register value and cannot be used to set a value to the actual SB register. To set an SB register value actually, write the following instruction immediately before or after this directive command. Example: LDC #80H,SB

- A symbol can be written in the operand.

### Description example

```
.SB      80H
LDC      #80,SB
```

## .SBBIT

Selects SB relative displacement addressing mode for bit symbol

### Function

- The 8-bit SB relative displacement addressing mode is selected by specifying a bit symbol of undefined value for the operand of this command.

### Description format

```
.SBBIT      (name)
.SBBIT      (name) [, (name)...]
```

### Rules for writing command

- Always be sure to enter a space or tab between the directive command and operand.
- A bit symbol defined by '.BTEQU' or '.BTGLB' can be written in the operand.
- A forward referenced bit symbol can be written in the operand.
- Before writing this directive command, be sure to set the SB register value by directive command ".SB".
- When specifying multiple names, separate them with a comma (,).

### Description example


```
.BTGLB      extbit
.SB         80H
LDC         #80H,SB
.SBBIT      bsym,extbit
BCLR        bsym          ;Select 8 bits SB
BAND        bsym          ;Select 8 bits SB
BSET        extbit        ;Select 8 bits SB
```



## .SBSYM

Selects SB relative displacement addressing mode

### Function

- The assembler selects the SB relative addressing mode for the name specified in the operand of this directive command.
  - The assembler selects the SB relative addressing mode for the expression in absolute 16-bit addressing mode that includes the name specified in the operand of this directive command.
  - The SB relative addressing mode can be selected for the operand that contains a relocatable value.
-  The SB relative addressing mode is not selected for the symbols that are defined by using the label name specified by this directive command.

### Description format

```
.SBSYM      (name)
.SBSYM      (name) [ , (name) . . . ]
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- A label and symbol can be written in the operand.
- Always be sure to set the SB register value with directive command ".SB" before you write this directive command.
- When specifying multiple names, be sure to separate the names with a comma as you write them.

**Description example**

```
.SB      80H  
LDC     #80H,SB  
.SBSYM  sym1,sym2
```

- In the following case, the SB relative addressing mode is not selected for sym2.

```
          .SBSYM  sym1  
sym2 .EQU  sym1+1
```

## .SECTION

Defines section name

### Function

- This command defines a section name.
- This command defines the beginning of a section. An interval from one section directive command to the next section directive command or directive command ".END" is defined as one section.
- This command defines a section type.
- If 'ALIGN' is specified, In308 allocates the beginning of a section to an even address.
- Directive command ".ALIGN" can be written in a ALIGN-specified section or an absolute-attribute section.

### Description format

```
.SECTION      (section name)
.SECTION      (section name),(section type)
.SECTION      (section name),(section type),ALIGN
.SECTION      (section name),ALIGN
```

### Rules for writing command

- Always be sure to write a section name when you define a section.
- When you write an assembly directive command to allocate a memory area or store data in memory or you write a mnemonic, always use this directive command to define a section.
- Write the section type and ALIGN after the section name.
- When specifying a section type and ALIGN, separate them with a comma as you write.
- Section type and ALIGN can be specified in any desired order.
- Section type can be selected from 'CODE', 'ROMDATA', and 'DATA'.
- The section type can be omitted. In this case, as308 assumes section type CODE as it processes assembling.

### Description example

```
.SECTION    program, CODE
NOP
.SECTION    ram, DATA
.BLKB 10
.SECTION    dname, ROMDATA
.BYTE "abcd"
.END
```

## .SJMP

Controls generation of a short-jump instruction

### Function

- This command controls generation of a short-jump instruction.
- No short-jump instruction is generated in lines after one in which ".SJMP OFF" is written.
- Short-jump instructions are generated in lines after one in which ".SJMP ON" is written.

### Description format

```
.SJMP ON
.SJMP OFF
```

### Description rules

- Be sure to insert a space or tab between this directive command and 'ON' or 'OFF.'

### Description example

```

:
.SJMP ON ; Generation of short jump is enabled.
JMP lab
NOP
.SJMP OFF
JMP lab ; Generation of short jump is disabled.
NOP
lab:
:
```

## .SUBSTR

Extracts specified number of characters

### Function

- This command extracts a specified number of characters from the specified position of a character string.
- 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.

### Description format

```
.SUBSTR {"(CS)",(ES),(NC)}  
.SUBSTR {'(CS)',(ES),(NC)}  
CS=character string  
ES=extract start position  
NC=number of characters to be extract
```

### Rules for writing command

- 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.
  - Separate the character string, extract start position, and the number of characters to be extracted with commas as you write them.
  - A symbol can be written in the 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.
- Kanji and other 8-bit code are not processed correctly. However, the as308 assembler does not output errors.
- Always be sure to enclose the character string with quotations as you write it.
- 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.

### Description example

```
name .MACRO      data
    .MREPEAT     .LEN{'data'}
    .BYTE .SUBSTR{'data',..MACREP,1}
    .ENDR
    .ENDM
    :
name  ABCD
    :
    .BYTE "A"
    .BYTE "B"
    .BYTE "C"
    .BYTE "D"
```

- 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.



## .VER

Transfers specified information to map file

### Function

- This command outputs the specified character string to a relocatable module file so it will be output to a map file when it is generated by ln308.
- All of the specified character strings are output to a map file.
- The user-specified information can be output to a map file for each relocatable module file.

### Description format

```
.VER      "(character string)"  
.VER      '(character string)'
```

### Rules for writing command

- Always be sure to insert space or tab between the directive command and the operand.
- Write the character string in the operand that you want to be output to a map file after enclosing it with single quotations (') or double quotations (").
- Make sure that the operand is written within the range of one line.
- This command can be written only once in one assembly source file.
- This command can be written in any desired line providing that it is entered before directive command ".END".

### Description example

```
.VER 'strings'  
.VER "strings"
```

## .WORD

Stores data in ROM in 2-byte length


### Function

- This command stores 2-byte long fixed data in ROM.
- Label can be defined at the address where data is stored.

### Description format

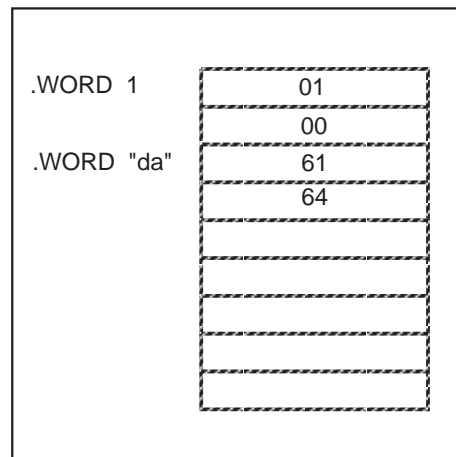
```
.WORD      (numeric value)
(name:)    .WORD      (numeric value)
```

### Rules for writing command

- Write an integral value in the operand.
  - Always be sure to insert space or tab between the directive command and the operand.
  - A symbol can be written in the operand.
  - An expression can be written in the operand.
  - When writing multiple operands, separate them with a comma (,).
  - A character or a string of characters can be written in the operand after enclosing it with single quotations (') or double quotations ("). In this case, data is stored in ASCII code representing the characters.
-  The length of a character string you can write in the operand is less than two characters.
- When defining a label, be sure to write the label name before the directive command.
  - Always be sure to insert a colon (:) after the label name.

### Description example

```
.SECTION    value,ROMDATA
.WORD 1
.WORD "da", "ta"
.WORD symbol
.WORD symbol+1
.WORD 1,2,3,4,5
.END
```



?

## Temporary label

**Function**

- This command defines a temporary label.
- The assembler references a temporary label that is defined immediately before or after an instruction.
- The labels that can be referenced are only the label defined 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 "tI0001", "tI0002" ... and "tIFFFF".

**Description format**

```
?:
(mnemonic)    ?+
(mnemonic)    ?-
```

### Rules for writing command

- 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.

### Description example

```
?:
    JMP  ?+
    JMP  ?-
?:
    JMP  ?-
```

Denotes a temporary label indicated by the arrow.

```
?:
    JMP  ?+
    JMP  ?-
?:
    JMP  ?-
```

## @

Concatenates character strings

### Function

- This command concatenates macro arguments, macro variables, reserved symbols, expanded file name of directive command "..FILE", and specified character strings.

### Description format

```
(character string) @ (character string)  
(character string) @ (character string) [@ (character string)...]
```

### Rules for writing command

- 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.
- If you want a concatenated character string to be a name, do not insert spaces and tabs before and after this directive command.

### Description example

```
.ASSERT "sample" > ..FILE@.dat
```

If the currently processed file name is "sample1.a30", a message is output to the sample.dat file.

- 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]
                 :
MOV LH         R0L,[A0]
```

# Technical Support Communication Sheet

Date : \_\_\_ / \_\_\_ / \_\_\_ ( Total Pages    )

**To Distributor:**

| <b>Contact Address</b> | <b>Product Information</b> |
|------------------------|----------------------------|
| Company :              | Product name :             |
| Department :           | Version number :           |
| Responsible person :   | Serial number :            |
| Phone :                | Host Machine name :        |
| FAX :                  | OS name :                  |
| E-mail :               | OS version :               |
| Address :              |                            |
| Message :              |                            |

If this form does not have sufficient space, use another sheet of paper to write your information.

( 1 /    )



# **AS308 V.1.00 User's Manual**

---

Second Edition: April 1, 1999

Document No. MSD-AS308-UE-990401

©1999 MITSUBISHI ELECTRIC CORPORATION

©1999 MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION

**MITSUBISHI ELECTRIC CORPORATION**

**MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION**

**RENESAS**  
Renesas Technology Corp.